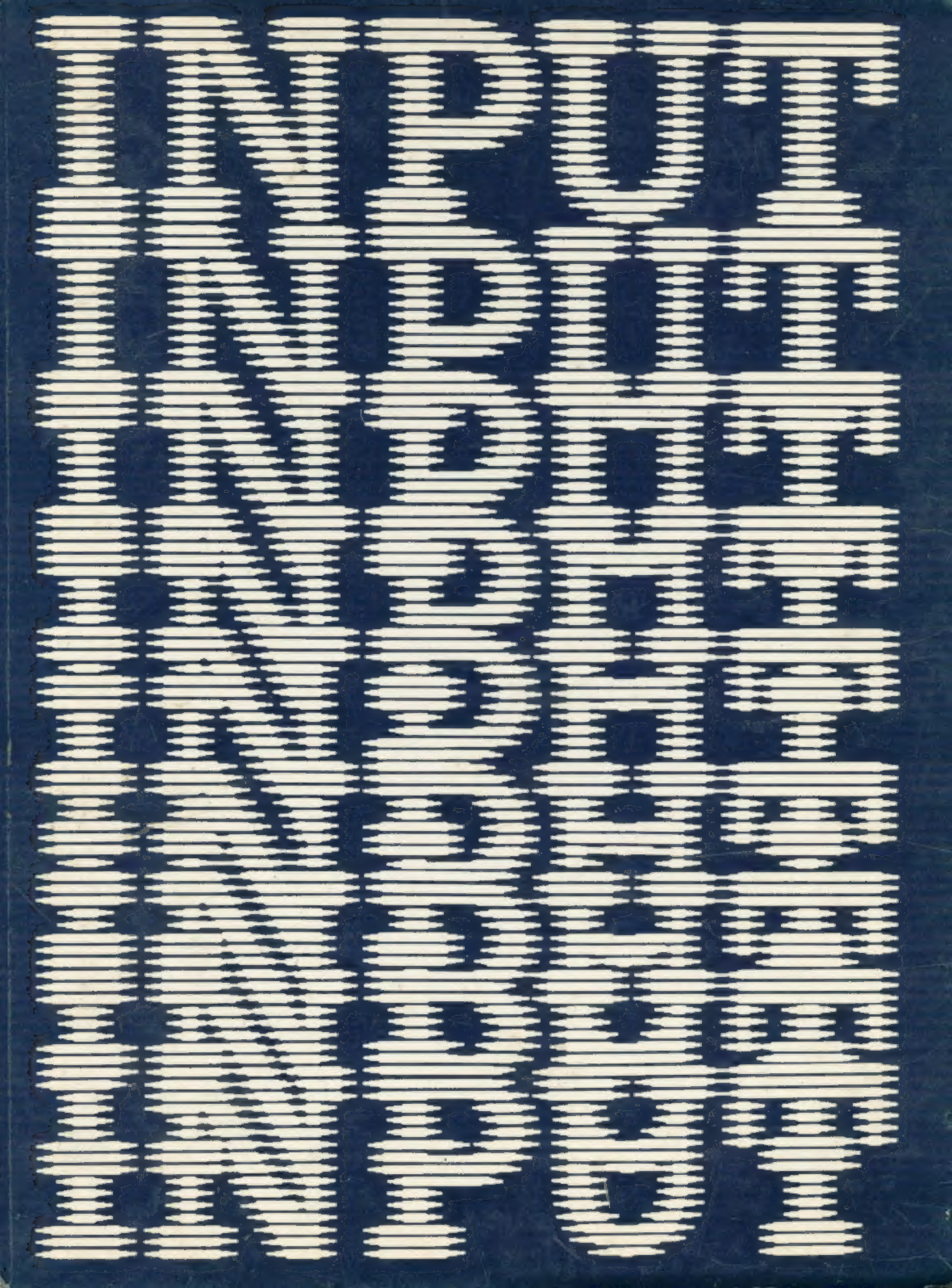

INPUT

INPUT

4

Editor
VICTOR CIVITA

REDAÇÃO

Diretor Editorial: Carmo Chagas

Editores Executivos:
Antonio José Filho, Berta Sztark Amar

Editor Chefe: Paulo de Almeida
Editores de texto: Ana Lúcia B. de Lucena,
Cláudio A. V. Cavalcanti
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Dagmar Bastos Sampaio,
Grace Alonso Arruda, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström,
José Benedito de Oliveira Damião, Maria de Lourdes Carvalho,
Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini
(Diretor do Núcleo de Informática Biomédica
da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria
em Informática Ltda., Campinas, SP
Tradução, adaptação, programação e redação:
Abílio Pedro Neto, Aluísio J. Dornellas de Barros,
Marcelo R. Pires Therezo, Marcos Huascar Velasco,
Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira
Coordenação Geral: Rejane Felizatti Sabbatini

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Maculan
Gerente de Circulação: Denise Maria Mozol

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, n.º 2000 - 3.º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda.
e impressa na Divisão Gráfica da Editora Abril S.A.

SUMÁRIO

APLICAÇÕES

CÓDIGO DE MÁQUINA

PERIFÉRICOS

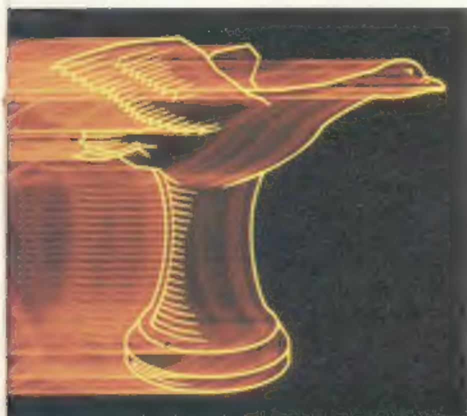
PROGRAMAÇÃO BASIC

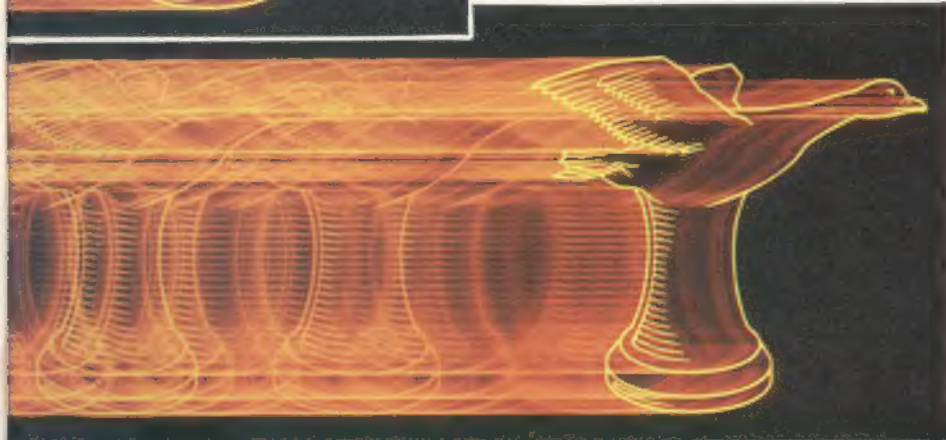
PROGRAMAÇÃO DE JOGOS

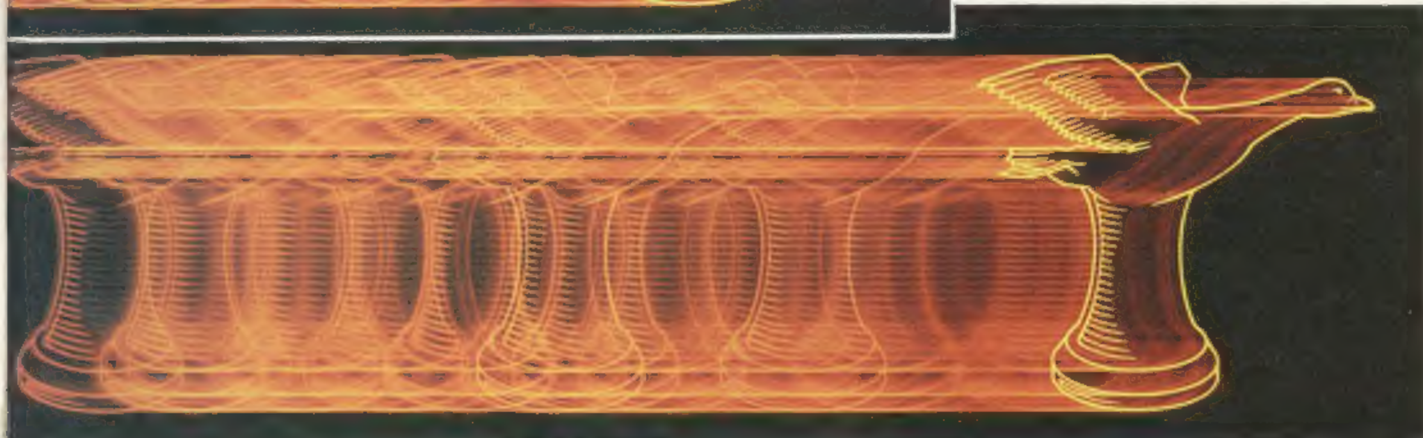
O JOGO A RAPOSA E OS GANSOS (2)

- FUNCIONAMENTO DO PROGRAMA
- INICIALIZAÇÃO
- COMECE O JOGO
- MAPEAMENTO DAS JOGADAS
- MAIS UMA VEZ?

Examinamos, no artigo anterior, os princípios que fundamentam o jogo A Raposa e os Gansos. Começaremos agora a montar o programa, digitando suas primeiras rotinas — entre elas, a que mapeia os movimentos.







Apresentamos aqui as rotinas de inicialização e de mapeamento dos movimentos. Além delas, você deverá digitar a rotina que oferece ao jogador mais uma partida. Nesse ponto, porém, a execução do programa não resultará em nada, pois ainda faltam rotinas muito importantes. Todas elas serão dadas no próximo artigo da série.

UMA VISÃO GERAL

O programa avalia as posições ocupadas no tabuleiro segundo a configuração das peças. Como cada posição tem um valor, pode-se, ao analisar as jogadas à frente, escolher o melhor movimento pelo resultado numérico maior.

O programa trabalha de três maneiras quando está analisando jogadas. A mais simples (nível 1) consiste no exame de apenas uma jogada. Nos níveis mais elevados, o programa examina as múltiplas possibilidades de jogo, utilizando o algoritmo alfa-beta para economizar tempo. Nos níveis intermediários, analisa todas as possibilidades abertas no momento.

As rotinas situadas da linha 2010 à linha 3000, que são executadas apenas uma vez, foram colocadas no fim do programa. No começo estão as rotinas mais importantes, o que garante uma maior velocidade ao jogo.

INICIALIZAÇÃO

As rotinas apresentadas a seguir, usadas para inicializar o jogo, dimensionam as matrizes, definem as funções e, no caso de alguns micros, estabelecem também o tabuleiro.

```
2010 DIM G(4): DEF FN U(A)=INT
(A-4*INT (A/4)): DEF FN V(A)=IN
T (A-8*INT (A/8))>=4: DEF FN W(
A)=INT (A-2*INT (A/2))
2015 LET HF=0: LET HG=0
2020 DIM B(32): LET B(1)=1: FOR
 I=1 TO 31: LET B(I+1)=B(I)*2:
NEXT I
2026 LET BX=B(32)*2-B(25): LET
E=1E30: LET H=-1E30
2030 LET L2=LN (2)
2040 DIM B$(16,2,16): DIM G$(2,
4): LET G$(1)="  ": LET G$(2)="
  "+CHR$ 146+CHR$ 147: DIM H$(2
,4): LET H$(1)="  ": LET H$(2)=
```

```
CHR$ 146+CHR$ 147+"  "
2050 LET X=1: FOR A=1 TO 2: FOR
 B=1 TO 2: FOR C=1 TO 2: FOR D=
1 TO 2: LET B$(X,1)=G$(D)+G$(C)
+G$(B)+G$(A)
2060 LET B$(X,2)=H$(A)+H$(B)+H$
(C)+H$(D): LET X=X+1: NEXT D: N
EXT C: NEXT B: NEXT A
2070 DIM S$(8,16): GOSUB 6000
2090 DIM F$(2,4): LET F$(1)="
"+CHR$ 144+CHR$ 145: LET F$(2)=
CHR$ 144+CHR$ 145+"  "
2095 DEF FN C(B)=FN U(B-1)*(4-8
*FN V(B-1))+12*FN V(B-1)
```



```
10 GOTO 2010
2010 DIM G(4),B(31),M(3,31),X(3
1),Z(31)
2020 B(0)=1:FOR K=1 TO 31:B(K)=
B(K-1)*2:NEXT
2026 BX=B(31)*2-B(24):E=1E30:H=
-1E30
2030 L2=LOG(2):DEFFNA(F)=INT(LO
G(F)/L2+.001)
```



```
7 KEYOFF:GOTO 2010
10 GOSUB 4000:GOTO 1010
2010 DIM G(4),B(31),M(3,31),X(3
1),Z(31),G$(4,1)
2020 B(0)=1:FOR K=1 TO 31:B(K)=
B(K-1)*2:NEXT
2026 BX=B(31)*2-B(24):E=1E+30:H
=-1E+30
2030 L2=LOG(2):DEFFNA(F)=INT(LO
G(F)/L2+1E-03)
2040 OPEN "GRP:" FOR OUTPUT AS
#1
```



```
10  GOTO 2010

2010  DIM G(4),B(31),M(3,31),X(
31),Z(31):V$ =  CHR$ (7) +  CHR
$ (1) +  CHR$ (1) +  CHR$ (7)
2020 B(0) = 1: FOR K = 1 TO 31:
B(K) = B(K - 1) * 2: NEXT
2026 BX = B(31) * 2 - B(24):E =
 1E30:H =  - 1E30
2030 L2 =  LOG (2): DEF  FN A(F
) =  INT ( LOG (F) / L2 + .001)
```

A linha 2010 dimensiona a matriz utilizada para armazenar as posições dos gansos. A linha 2020 numera cada quadrado do tabuleiro.

A linha 2030 determina o número total de configurações que o programa pode avaliar. O valor 0.001 foi adicionado à definição da função A para evitar a ocorrência de erros de arredondamento no cálculo de logaritmos.

No Spectrum, a matriz B$, dimensionada na linha 2040, mostra as colunas do tabuleiro já ocupadas pelas peças. F$ é usada para mostrar a peça da raposa e a casa em que está, e H$, para os gansos e suas casas. A matriz S$, determina o número dos quadrados.

Nos demais microcomputadores, as figuras são definidas em alta resolução por uma outra rotina.

COMEÇANDO

Esta rotina permite a escolha das peças com que se vai jogar e o grau de dificuldade. Comece pelo mais fácil:



```
2700 LET F=2: LET G(1)=29: LET
G(2)=30: LET G(3)=31: LET G(4)=
32: GOSUB 2710: GOTO 1010
2710 CLS : PRINT AT 0,8; INK 1;
"RAPOSA E GANSOS": INPUT "VOCE
QUER ...";TAB 5;"SER A RAPOSA ?
(S/N)    ";I$
2720 LET PF=0: IF I$="S" OR Y$=
"s" THEN  GOTO 2760
2730 LET PF=1: IF I$<>"N" AND I
$<>"n" THEN  GOTO 2710
2740 INPUT "NIVEL DE HABILIDADE
 DA RAPOSA ? ";SF: IF SF<1 OR S
F>10 THEN  GOTO 2740
2750 LET HF=131*(SF=5)+613*(SF=
6)+1997*(SF>6)
2760 INPUT "VOCE QUER ...";TAB
5;"CONTROLAR OS GANSOS ?(S/N) "
;I$
2770 LET PG=0: IF I$="S" OR I$=
"s" THEN  GOTO 2860
2780 LET PG=1: IF I$<>"N" AND I
$<>"n" THEN  GOTO 2760
2790 INPUT "NIVEL DA HABILIDADE
 DOS GANSOS  ";SG: IF SG<1 OR S
G>10 THEN  GOTO 2790
2800 LET HG=131*(SG=5)+613*(SG=
6)+1997*(SG>6): IF HF<HG THEN
LET HF=HG
2860 INPUT "VOCE QUER ALTERAR A
S POSICOES   INICIAIS ?";I$: IF
 I$="N" OR I$="n" THEN  GOTO 30
00
2880 IF I$<>"S" AND I$<>"s" THE
N  GOTO 2860
2890 GOSUB 210: GOSUB 310: INPU
T "QUER MOVER A RAPOSA ?";I$
2900 IF I$="N" OR I$="n" THEN
GOTO 2930
2910 IF I$<>"S" AND I$<>"s" THE
N  GOTO 2890
2920 INPUT "PARA ONDE ?";F: IF
F<1 OR F>32 THEN  GOTO 2920
2930 FOR G=1 TO 4: GOSUB 210: G
OSUB 310
2940 INPUT "QUER MOVER O GANSO
DA POSICAO";(G(G));"?";I$
2950 IF I$="N" OR I$="n" THEN
GOTO 2990
2960 IF I$<>"S" AND I$<>"s" THE
N  GOTO 2940
2970 INPUT "PARA ONDE ?";I: IF
FN X(I) OR I=F THEN  GOTO 2960
2972 IF I<1 OR I>32 THEN  GOTO
2970
2980 LET G(G)=I
2990 NEXT G: IF FN X(F) THEN  P
RINT "HA UM GANSO SOB A RAPOSA"
: FOR I=1 TO 1500: NEXT I: GOTO
 2910
3000 RETURN
```





```
2500 DIM R(1400),S(1400)
2700 F=1:G(1)=28:G(2)=29:G(3)=3
0:G(4)=31:GOSUB 2710:GOTO 1010
2710 CLS:PRINT "VOCE QUER SER A
 RAPOSA (S/N) ?";
2720 K$=INKEY$:IF K$<>"S" AND K
$<>"N" THEN 2720
2730 PRINT K$:PF=1:IF K$="S" TH
EN PF=0:GOTO 2760
2740 PRINT:PRINT "NIVEL DE HABI
LIDADE DA RAPOSA   (0-9) ?";
2745 K$=INKEY$:IF K$<"0" OR K$>
"9" THEN 2745
```

```
2746 SF=VAL(K$)+1:PRINT K$
2750 HF=-131*(SF=5)-613*(SF=6)-
1399*(SF>6)
2760 PRINT:PRINT "VOCE QUER CON
TROLAR OS GANSOS   (S/N) ?";
2770 K$=INKEY$:IF K$<>"S" AND K
$<>"N" THEN 2770
2780 PRINT K$:PG=1:IF K$="S" TH
EN PG=0:GOTO 2860
2790 PRINT:PRINT"NIVEL DE HABIL
IDADE DOS GANSOS  (0-9) ?";
2795 K$=INKEY$:IF K$<"0" OR K$>
"9" THEN 2795
2796 SG=VAL(K$)+1:PRINT K$
2800 HG=-131*(SG=5)-613*(SG=6)-
1399*(SG>6):IF HF<HG THEN HF=HG
2860 PRINT:PRINT"VOCE QUER ALTE
RAR AS POSICOES   INICIAIS (S/N
)?";
2870 K$=INKEY$:IF K$<>"S" AND K
$<>"N" THEN 2870
2880 IF K$="N" THEN 3000
2890 GOSUB 210
2920 DRAW"BM180,80"+MW$:XX=FNXX
(1):YY=FNYY(1):GOSUB 1810:F=4*I
NT(YY/20):F=FNCN(F)
2925 PUT(68,8)-(87,27),SQ,PSET:
PUT(XX,YY+5)-(XX+19,YY+13),FX,P
SET
2930 FOR G=1 TO 4:GOSUB 210
2940 XX=FNXX(G(G)):X1=XX:YY=FNY
Y(G(G)):Y1=YY:GOSUB 1810:PUT(X1
,Y1)-(X1+19,Y1+19),SQ,PSET
2950 I=4*INT(YY/20):I=FNCN(I)
2960 IF (FNX(I) OR I=F) AND I<>
G(G) GOSUB 5000:GOTO 2940
2970 PUT(XX,YY+5)-(XX+19,YY+14)
,GS,PSET:G(G)=I
2990 NEXT:IF FNX(F) GOSUB 5000:
GOTO 2920
2995 C=1:G=G(1)
3000 RETURN
```





```
2500 DIM R(1100),S(1100)
2700 F=1:G(1)=28:G(2)=29:G(3)=3
0:G(4)=31:GOSUB 2710:GOTO 10
2710 CLS:PRINT"Alguém joga com
a raposa? (S/N) ";
2720 K$=INKEY$:IF K$<>"S" AND K
$<>"N" THEN 2720
2730 PRINTK$:PF=1:IF K$="S" THE
N PF=0:GOTO 2760
2740 PRINT:PRINT"Habilidade da
raposa? (0-9) ";
2745 K$=INKEY$:IF K$<"0" OR K$>
"9" THEN 2745
2746 SF=VAL(K$)+1:PRINTK$
2750 HF=-131*(SF=5)-613*(SF=6)-
1099*(SF>6)
2760 PRINT:PRINT"Alguém joga co
m os gansos? (S/N) ";
2770 K$=INKEY$:IF K$<>"S" AND K
$<>"N" THEN 2770
2780 PRINTK$:PG=1:IF K$="S" THE
N PG=0:GOTO 2860
2790 PRINT:PRINT"Habilidade dos
 gansos? (0-9) ";
2795 K$=INKEY$:IF K$<"0" OR K$>
"9" THEN 2795
2796 SG=VAL(K$)+1:PRINTK$
2800 HG=-131*(SG=5)-613*(SG=6)-
1099*(SG>6):IF HF<HG THEN HF=HG
2860 PRINT:PRINT"Você quer alte
rar as posições inicais?  (S/N)
 ";
2870 K$=INKEY$:IF K$<>"S" AND K
$<>"N" THEN 2870
2880 IF K$="N" THEN 3000
2890 GOSUB 210:GOSUB 4000:GOSUB
2920:GOTO 1010
2920 PRESET (188,80):PRINT#1,"P
ara ?":XX=FNXX(1):YY=FNYY(1):GO
SUB 1810:F=4*INT((YY-2)/20):F=F
NCN(F)
2925 PUT SPRITE FX,(XX,YY),6
2930 FOR G=1 TO 4:GOSUB 210
2940 XX=FNXX(G(G)):YY=FNYY(G(G)
):GOSUB 1810
2950 I=4*INT((YY-2)/20):I=FNCN(
I)
2960 IF (FNX(I) OR I=F) AND I<>
G(G) THEN GOSUB 5000:GOTO 2940
2970 GS(5-G,1)=I:G(G)=I:PUT SPR
ITE GS(5-G,0),(XX,YY),15
2990 NEXT:IF FNX(F) THEN GOSUB
5000:GOTO 2920
2995 C=1:G=G(1)
3000 RETURN
```



```
2500  DIM R(1500),S(1500)
2700 F = 1: FOR I = 28 TO 31:G(
I - 27) = I: NEXT : GOSUB 2710:
 GOTO 1010
2710  HOME : VTAB 22: PRINT "AL
GUEM JOGA COM A RAPOSA? (S/N) "
;
2720  GET K$: IF K$ <  > "S" AN
D K$ <  > "N" THEN 2720
2730  PRINT K$:PF = 1: IF K$ =
"S" THEN PF = 0: GOTO 2760
2740  PRINT "HABILIDADE DA RAPO
SA? (0-9) ";
2745  GET K$: IF K$ < "0" OR K$
 > "9" THEN 2745
2746 SF =  VAL (K$) + 1: PRINT
K$
2750 HF = 131 * (SF = 5) + 613
* (SF = 6) + 1499 * (SF > 6)
2760  HOME : VTAB 22: PRINT "AL
GUEM JOGA COM OS GANSOS? (S/N)
";
2770  GET K$: IF K$ <  > "S" AN
D K$ <  > "N" THEN 2770
2780  PRINT K$:PG = 1: IF K$ =
"S" THEN PG = 0: GOTO 2860
2790  PRINT "HABILIDADE DOS GAN
SOS? (0-9) ";
2795  GET K$: IF K$ < "0" OR K$
 > "9" THEN 2795
2796 SG =  VAL (K$) + 1: PRINT
K$
2800 HG = 131 * (SG = 5) + 613
* (SG = 6) + 1499 * (SG > 6): I
F HF < HG THEN HF = HG
2860  HOME : VTAB 22: PRINT "QU
ER MUDAR AS POSICOES INICIAIS?
(S/N) ";
2870  GET K$: IF K$ <  > "S" AN
D K$ <  > "N" THEN 2870
2880  IF K$ = "N" THEN 3000
2890  GOSUB 210
2920  HOME : VTAB 22: PRINT "MO
VE PARA ONDE?":XX =  FN XX(1):Y
Y =  FN YY(1): GOSUB 1810:F = 4
 *  INT (YY / 20):F =  FN CN(F)

2925  HCOLOR= 0: DRAW FX AT 120
,8: HCOLOR= 3: DRAW FX AT XX +
2,YY + 8
2930  FOR G = 1 TO 4: GOSUB 210

2940 :XX =  FN XX(G(G)):XN = XX
:YY =  FN YY(G(G)):YN = YY: GOS
UB 1810: HCOLOR= 0: DRAW GS AT
XN + 7,YN + 5
2950 I = 4 *  INT (YY / 20):I =
 FN CN(I)
2960  IF ( FN X(I) OR I = F) AN
D I <  > G(G) THEN  GOSUB 5000:
 GOTO 2920
2970  HCOLOR= 3: DRAW GS AT XX
+ 7,YY + 5:G(G) = I
2990  NEXT : IF  FN X(F) THEN
GOSUB 5000: GOTO 2920
2995 C = 1:G = G(1)
3000  RETURN
```

A linha 2700 determina as posições iniciais, com os quatro gansos ocupando os quatro quadrados da coluna inferior do tabuleiro, e a raposa, o segundo quadrado a partir da esquerda do vídeo, na coluna superior.

Em seguida, as linhas 2710 a 2750 pedem que se defina quem jogará pela raposa e que seja indicado um nível de dificuldade de 1 a 10, se for o computador. As linhas 2760 a 2800 são semelhantes, só que tratam dos gansos.

O jogador pode ajustar as posições iniciais não só para dar continuidade a uma partida (será preciso tomar nota das posições), mas, também, para estudar e tentar ganhar o jogo de uma posição particularmente interessante. As linhas 2860 a 3000 perguntam se o jogador quer alterar as posições de início, realiza todas as modificações necessárias e verifica se estas estão de acordo com as regras.

MAPEAMENTO DAS JOGADAS

A rotina de mapeamento de jogadas é uma das mais importantes do jogo:

S

```
140 DEF FN X(B)=B=G(1) OR B=G(
2) OR B=G(3) OR B=G(4)
2100 DIM R$(8,16)
2142 DEF FN Z(B)=(B=G(1))+(B=G(
2))*2+(B=G(3))*3+(B+G(4))*4
2150 DIM M(4,32): DIM X(32): DI
M Z(32)
2160 FOR B=1 TO 32: LET U=B-1-4
*INT (B/4-.2): FOR A=1 TO 4: LE
T M(A,B)=(B-2)-2*U+8*((B<5) OR
(A>2))+(A*7-6)*(U=3)+(A=2)+(A=2
)+(A=4): NEXT A: LET X(B)=((B>4
)+(B<29))*((U<3)+1): LET Z(B)=(
B>4)*((U<3)+1): NEXT B
2180 DIM V(11): DIM A(11): DIM
F(11): DIM P(11): DIM C(11): DI
M R(1): DIM S(1)
```

T

```
2110 DEFFNF(B)=((B>3)+(B<28))*(
((3ANDB)<3)-1)-1
2120 DEFFNG(B)=(B>3)*(((3 AND B
)<3)-1)-1
2140 DEFFNX(B)=(B=G(1)ORB=G(2)O
RB=G(3)ORB=G(4))
2142 DEFFNZ(B)=-(B=G(1))-(B=G(2
))*2-(B=G(3))*3-(B=G(4))*4
2150 DEFFNXX(B)=-((7ANDB)<4)*(2
8+40*(3ANDB))-((7ANDB)>3)*(128-
40*(3ANDB))
2155 DEFFNYY(B)=B+20*INT(B/4)
2156 DEFFNCN(B)=B-((7ANDB)<4)*(
XX-28)/40-((7ANDB)>3)*(128-XX)/
40
2160 FOR B=0 TO 31:FOR A=0 TO 3
:M(A,B)=B-2*(3ANDB)-2-8*(B<4 OR
 A>1)-(1+A*7)*((3 AND B)=3)+(1
AND A):NEXT:X(B)=FNF(B):Z(B)=FN
G(B):NEXT
```

```
2110 DEFFNF(B)=((B>3)+(B<28))*(
((BMOD4)<3)-1)-1
```



```
2120 DEFFNG(B)=(B>3)*(((BMOD4)<
3)-1)-1
2140 DEFFNX(B)=(B=G(1)ORB=G(2)O
RB=G(3)ORB=G(4))
2142 DEFFNZ(B)=-(B=G(1))-(B=G(2
))*2-(B=G(3))*3-(B=G(4))*4
2144 DEFFNZZ(B)=-(B=GS(1,1))-(B
=GS(2,1))*2-(B=GS(3,1))*3-(B=GS
(4,1))*4
2150 DEFFNXX(B)=-((BMOD8)<4)*(3
8+40*(BMOD4))-((BMOD8)>3)*(138-
40*(BMOD4))
2155 DEFFNYY(B)=20+20*INT(B/4)
2156 DEFFNCN(B)=B-((BMOD8)<4)*(
XX-38)/40-((BMOD8)>3)*(138-XX)/
40
2160 FOR B=0 TO 31:FOR A=0 TO 3
:M(A,B)=B-2*(BMOD4)-2-8*(B<4 OR
 A>1)-(1+A*7)*((BMOD4)=3)+(AMOD
2):NEXT:X(B)=FNF(B):Z(B)=FNG(B)
:NEXT
```



```
2110  DEF  FN MD(X) = X - ( INT
 (X / MD) * MD): DEF  FN M2(X)
= X - ( INT (X / 2) * 2): DEF
FN M4(X) = X - ( INT (X / 4) *
4): DEF  FN M8(X) = X - ( INT (
X / 8) * 8)
2115  DEF  FN F(B) = ((B > 3) +
 (B < 28)) * (( FN M4(B) < 3) +
 1) - 1
2120  DEF  FN G(B) = (B > 3) *
(( FN M4(B) < 3) + 1) - 1
2140  DEF  FN X(B) =  - (B = G(
1) OR B = G(2) OR B = G(3) OR B
 = G(4))
2142  DEF  FN Z(B) = (B = G(1))
 + (B = G(2)) * 2 + (B = G(3))
 * 3 + (B = G(4)) * 4
2150  DEF  FN XX(B) = ( FN M8(B
) < 4) * (X1 + 21 + 40 *  FN M4
(B)) + ( FN M8(B) > 3) * (X2 -
41 - 40 *  FN M4(B))
2155  DEF  FN YY(B) = Y1 + 20 *
  INT (B / 4)
2156  DEF  FN CN(B) = B + ( FN
M8(B) < 4) * (XX - 78) / 40 + (
 FN M8(B) > 3) * (178 - XX) / 4
0
2160  FOR B = 0 TO 31: FOR A =
0 TO 3:M(A,B) = B - 2 *  FN M4(
B) - 2 + 8 * (B < 4 OR A > 1) +
 (1 + A * 7) * ( FN M4(B) = 3)
+  FN M2(A): NEXT :X(B) =  FN F
(B):Z(B) =  FN G(B): NEXT
```

As linhas 2110 a 2160 montam o mapa dos movimentos da raposa e dos gansos na matriz M, armazenam o número dos movimentos possíveis da raposa na matriz X e o dos gansos na matriz Z. As matrizes são formadas pelas funções definidas nas linhas 2110 a 2142.

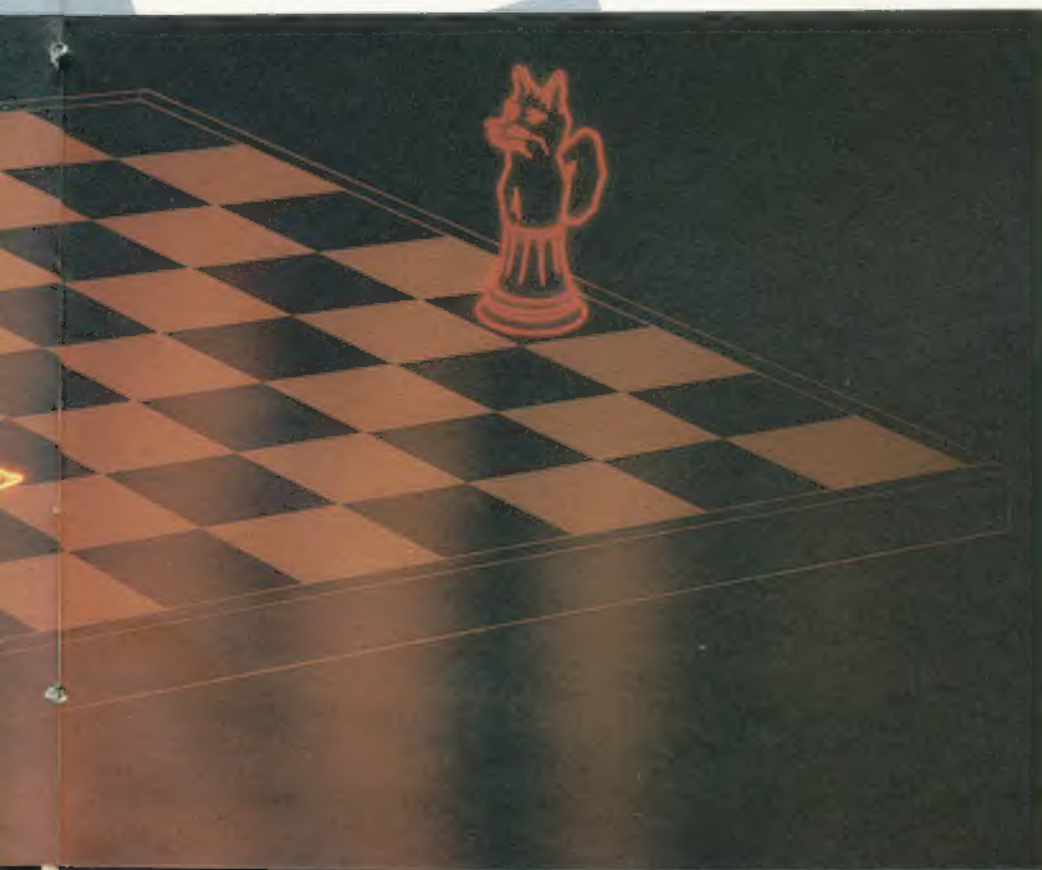
A rotina do Spectrum é mais curta devido à lógica interna dessa máquina.

MAIS UMA VEZ?

Agora, adicione esta rotina:



```
1410 INPUT "QUER JOGAR DE NOVO
(S/N) ?";I$
```





O que é Inteligência Artificial?

A Inteligência Artificial (IA), cujos conceitos são utilizados no jogo *A Raposa e os Gansos*, é um campo da Informática que procura desenvolver métodos computacionais para imitar o intelecto humano em várias tarefas, como o diagnóstico de doenças, a prospecção mineral etc. Os jogos estratégicos, como o xadrez, também têm sido alvo de estudos. Aplicam-se a eles sobretudo duas técnicas de IA: a *programação heurística* e a *otimização de busca.*

```
1420 IF I$="S" OR I$="s" THEN
GOTO 2700
1430 IF I$<>"N" AND I$<>"n" THE
N  GOTO 1410
1440 STOP
```



```
1410 PRINT @390,"QUER RECOMECAR
 (S/N) ?"
1420 K$=INKEY$:IF K$="S" GOSUB
4040:CLS:GOTO 2700
1430 IF K$<>"N" THEN 1420
1440 CLS:END
```



```
1410 LOCATE 5,20:PRINT"QUER REC
OMEÇAR? (S/N)".
1420 K$=INKEY$:IF K$="S" THEN R
UN
1430 IF K$<>"N" THEN 1420
1440 CLS:END
```



```
1410  FOR Z = 1 TO 5000: NEXT :
 TEXT : HOME : PRINT "JOGA OUTR
A VEZ? (S/N) ";
1420  GET K$: IF K$ <  > "S" AN
D K$ <  > "N" THEN 1420
1430  IF K$ = "S" THEN  RUN
1440  HOME : END
```

Essas linhas entrarão em cena quando os gansos conseguirem encurralar a raposa ou quando a raposa conseguir atingir o lado oposto do tabuleiro.

Não tente nesta fase executar o programa, pois ainda falta adicionar várias partes vitais. Com as rotinas do próximo artigo você poderá, finalmente, começar a jogar.

COMO UTILIZAR UM DISQUETE

O emprego de um acionador de disquetes em seu microcomputador pode envolver algumas complicações. Este artigo mostra como evitar erros comuns e como fazer o melhor uso desse periférico.

Como vimos no artigo da página 876, o disco magnético é a melhor alternativa para quem deseja um armazenamento mais rápido e confiável do que o proporcionado pela fita cassete. Entretanto, ao contrário do que ocorre com o gravador de fita, pode ser complicado fazer funcionar a combinação computador-disquete. Além disso, o usuário precisa aprender uma série de novos comandos para controlar o disco.

Este artigo explica como conectar e usar os acionadores de disquetes disponíveis para os diversos tipos de microcomputador existentes no Brasil. As linhas cobertas no artigo são: TRS-80, TRS-Color, Apple II, TK-2000 e MSX. Uma breve menção é feita aos micros compatíveis com a linha Spectrum (TK-90X), que dispõe de unidades acionadoras apenas no Exterior.

A primeira coisa que você deve fazer ao receber sua unidade de disquetes é checar se ela foi protegida para transporte. Em geral, um pedaço de cartolina do tamanho de um disquete é inserido na unidade, a fim de proteger a cabeça de leitura e gravação. Retire esse protetor somente depois de colocar o acionador no local em que será utilizado. Guarde-o, porém, para que, quando se fizer necessário, possa mudar o computador de lugar.

COMO CONECTAR

Conectar a unidade acionadora de disquetes ao micro pode ser fácil ou complicado, dependendo do modelo.

Os microcomputadores compatíveis com a linha TRS-Color utilizam unidades acionadoras de disquetes de 5 1/4 polegadas (minidisquetes), de face simples e densidade dupla. Encontram-se no mercado gabinetes de um ou de dois acionadores (como os do CP-400).

A conexão é muito simples: o cabo de ligação, do tipo plano, tem na ponta um conector que deve ser inserido na porta de cartuchos do micro.

O software operacional de utilização já está gravado em memória ROM, na própria unidade de disco. Dessa maneira, torna-se instantaneamente disponível tão logo se liga o computador, não sendo necessário carregá-lo de um disquete. O cabo de força da unidade acionadora deve ser ligado separadamente em uma tomada.



Os microcomputadores compatíveis com a linha TRS-80 utilizam unidades acionadoras de disquetes de 5 1/4 polegadas (minidisquetes), de face simples ou dupla e densidade dupla. É possível concectar até quatro unidades por micro. Alguns modelos de computador comportam dois acionadores no gabinete principal e mais dois em caixas separadas. Outras marcas aceitam apenas unidades externas.

Em geral, os acionadores já vêm instalados nos micros que os aceitam no gabinete principal, quando se compra o modelo com disquetes. Se você possui um computador desse tipo e pretende adicionar um ou dois acionadores, recomendamos que não o faça sozinho: peça auxílio ao revendedor, pois a instalação é razoavelmente complexa e requer a abertura do gabinete.

Nos computadores em que os acionadores são ligados externamente, a conexão é muito simples: o cabo de ligação, do tipo plano, tem na ponta um conector que deve ser inserido na porta de expansão de discos do micro. Para ligar mais de uma unidade, deve-se utilizar um cabo próprio, com vários conectores intercalados (*daysy chaining*). Não são necessários cabos de força separados para as unidades acionadoras, pois a alimentação ocorre por intermédio do cabo de conexão.



Os microcomputadores que são compatíveis com a linha Apple II utilizam unidades acionadoras de disquetes de 5 1/4 polegadas (minidisquetes), de face simples e densidade dupla. A placa de controle (interface) deve ser adquirida separadamente, e podem ser conectadas a ela até duas unidades acionadoras. O número de placas de controle que cabem no micro depende da marca, mas, de um modo geral, empregam-se até três placas. Os acionadores costumam alojar-



se individualmente em caixas externas.

A conexão não é muito simples, porém com alguns cuidados, você mesmo pode fazê-la. Primeiro, desligue o micro da tomada e abra a tampa superior do console (UCP). Localize um conector interno vazio para colocar a placa de controle — utilizam-se, normalmente, os conectores 6 e/ou 5 para isso. O cabo de ligação da unidade, do tipo plano, tem na ponta um conector fêmea que deve ser inserido no macho correspondente, na placa de controle. Não são necessários cabos de força separados para as unidades acionadoras, pois a alimentação ocorre por intermédio do cabo de conexão.



O microcomputador TK 2000 pode usar apenas um acionador de disquetes, com discos de 5 1/4 polegadas (minidisquetes), de face simples e densidade dupla. O acionador é ligado ao console através de uma interface externa. Esta é adquirida à parte e deve ser inserida na porta de expansão do equipamento. Desligue o micro da tomada para proceder à conexão.

O cabo de força da interface e do acionador deve ser ligado separadamente na tomada, pois a UCP não tem potência suficiente para ambos.



Os microcomputadores compatíveis com a linha MSX I utilizam unidades acionadoras de disquetes de 5 1/4 polegadas (minidisquetes), de face simples e densidade dupla, ou microdisquetes de 3,5 polegadas, de face dupla (encontrados apenas no Exterior). Existem gabinetes de um ou de dois acionadores disponíveis no mercado.

A conexão é muito simples: o cabo de ligação tem a interface e um conector na ponta. Este deve ser inserido numa das portas de cartuchos do microcomputador. O software operacional de utilização já está gravado em memória ROM, na própria unidade de disco. Assim, torna-se instantaneamente disponível tão logo se liga o computador, não sendo necessário carregá-lo de um disquete. A unidade acionadora não requer cabo de força separado.



Não existem no mercado nacional unidades de disquete para os micros compatíveis com a linha Sinclair ZX Spectrum. Nos Estados Unidos e na Eu-

ropa, podem ser adquiridos facilmente dois tipos de sistemas de armazenamento magnético auxiliar para esses micros: os acionadores de disquetes de 5 1/4 polegadas, de face simples e densidade simples (em gabinetes com até dois acionadores, e descanso para o teclado), e os acionadores de fitas sem fim (*tape loop*). Estes, por suas características, aproximam-se dos disquetes em modo de operação, e custam bem mais barato. Os modelos mais conhecidos são o Microdrive, da própria Sinclair, e o Wafadrive. Podem ser ligados até oito acionadores em paralelo.

Para ligar o Microdrive, é necessário ter também uma Interface 1 da Sinclari, ou similar, que deve ser inserida na porta de expansão do teclado. Para fazê-lo, desligue o computador. O cabo de conexão, do tipo plano, tem duas pontas: uma para inserção na Interface e outra para o Microdrive. Cada Microdrive, por sua vez, tem um conector fêmea que possibilita a ligação de vários acionadores. Não são necessários cabos de força separados para os acionadores, pois os mesmos são alimentados pelo computador.

FORMATAÇÃO

Antes de usar o disquete para armazenar informação, deve-se prepará-lo por meio de um procedimento de software, chamado de formatação ou inicialização. O programa de formatação simplesmente impõe um padrão de trilhas e setores no disco virgem, definindo (isto é, gravando certos apontadores e listas) o modo como a informação será armazenada no disco.

Os disquetes são inicialmente divididos em trilhas concêntricas, por sua vez "fatiadas" em setores de igual número de bytes. O controlador de disquetes reconhece precisamente as marcas feitas pelo programa formatador. Esse procedimento é chamado de setoreamento por programa (*soft-sectoring*).

O processo de formatação varia conforme a marca do computador e o sistema operacional empregado. Eis aqui os comandos que você pode utilizar para formatar um disquete:



Coloque o disquete com o sistema operacional na unidade acionadora nº 0. Existem duas opções para a formatação de um disquete virgem nos micros desta linha: somente formatação - que produz um disquete sem o sistema operacional — e cópia total — que copia tudo o que o disco-mestre tem, inclusive o sistema operacional. A operação difere conforme o número de acionadores de disquetes do computador.

Para formatar, apenas, digite:

```
FORMAT
```

O programa fará, então, uma série de perguntas. Primeiro, pedirá que você indique a unidade em que está o disquete. Se você tem apenas um acionador, indique *drive 0* e retire o disquete com o sistema operacional do mesmo, colocando o disco virgem em seu lugar. Se você tem dois ou mais acionadores, não é preciso fazer isso: basta inserir o disquete virgem em outra unidade e indicar o seu número ao programa (por exemplo, *drive 1*).

Ao executar a formatação, o programa indicará o número da trilha que está formatando. Ao final, retire o disquete formatado da unidade.

No caso de formatação com cópia, use o comando:

```
BACKUP
```

Entre as informações solicitadas, o programa pede ao usuário que indique se deseja reformatar o disquete, caso ele já esteja previamente formatado. Se a resposta for sim, o programa funcionará como foi explicado anteriormente e, depois, fará a cópia do disquete-mestre. Se você tem apenas um acionador, precisará realizar várias vezes a operação de retirar disquete de cópia e inserir disquete-mestre. Fique atento para não trocar os discos. Convém proteger o disquete-mestre colocando uma etiqueta adesiva sobre o orifício quadrado da margem do envelope. Será preciso, ainda, saber a senha de acesso do disquete-mestre.



Coloque o disquete virgem na unidade acionadora, feche a porta da mesma e digite pelo teclado:

```
DSKINIT
```



Coloque o disquete com o sistema operacional na unidade acionadora 1. Proceda à inicialização, carregando o sistema operacional.

Tire o disquete do sistema e coloque o disquete virgem na unidade acionadora. Feche a porta da mesma e digite pelo teclado o comando **NEW**. Todos os disquetes com o sistema operacional precisam ter um programa em BASIC gravado. Este será carregado automaticamente toda vez que o computador for ligado. Escolha e digite o programa que desejar: ele pode conter desde uma linha de saudação até vários comandos que executem alguma tarefa específica. Em seguida, digite o comando:

```
INIT NOME
```

NOME refere-se ao programa inicial de carregamento, que normalmente recebe a designação de HELLO ou ALO. A formatação será, então, realizada.



Coloque o disquete com o sistema operacional na unidade acionadora A. A operação de formatação varia conforme o número de acionadores de disquete do computador.

Para formatar, apenas, digite:

```
FORMAT A·
```

se o seu computador tem apenas um acionador, ou:

```
FORMAT B·
```

se o seu computador tem dois acionadores.

O programa dará, então, algumas informações. Se você tem apenas um acionador, indique *drive A* e retire o disquete com o sistema operacional do mesmo, colocando o disco virgem em seu lugar. Se você tem dois ou mais acionadores, não é preciso fazer isso: basta inserir o disquete virgem em outra unidade e indicar o seu número ao programa (por exemplo, *drive B*).

Ao final, retire o disquete formatado da unidade.



Para a formatação de um Microdrive, use o comando:

```
FORMAT "M";1;"NOME"
```

NOMES DE ARQUIVOS

Tudo o que é armazenado em disquete — seja um programa, seja um conjunto de dados — precisa ter um nome. O sistema operacional de disco trata de forma igual dados e programas: ambos são chamados de *arquivos*. O número de caracteres permitidos no nome de um arquivo de disco varia segundo o modelo do computador. A maioria deles aceita nomes formados por oito caracteres, no máximo. Os micros da linha Apple, po-



rém, aceitam até trinta caracteres, e o Microdrive para o Spectrum, até dez. O Apple e, também, o único micro que permite a inclusão de espaços em branco e de outros caracteres especiais dentro de um nome de arquivo. Pode-se perfeitamente chamar um arquivo de DADOS DE ENTRADA, por exemplo. Os computadores das linhas TRS-80, TRS-Color e MSX, ao contrário, exigem que o nome do arquivo não tenha espaços em branco e que comece com um caractere alfabético.

Nesses computadores, quando se usa o comando **SAVE** para armazenar alguma coisa em disco, a máquina automaticamente adiciona um *sufixo* ao nome que foi fornecido. Esse sufixo pode ser **BAS**, que significa que um programa em BASIC foi gravado, **BAK**, que indica que o arquivo é uma cópia (**BAcK-up**) de outro arquivo no mesmo disco, ou **DAT**, que identifica um arquivo de dados, distinguindo-o de um programa. Conforme a linguagem, outros sufixos podem ser utilizados, mas devem ter sempre um máximo de três letras. Nos micros das linhas TRS-80 e TRS-Color, são separados do nome principal por uma barra (PROG/BAS, por exemplo); nos da linha MSX por um ponto (PROG. BAS, por exemplo).

Não existem sufixos nos nomes de arquivos para as linhas Apple e TK-2000.

Os nomes de arquivos são essenciais para a operação de um sistema que utiliza discos, pois cada um destes possui um catálogo, ou lista com todos os nomes de arquivos gravados. Tal catálogo (*catalog* ou *directory*) é atualizado sempre que um novo arquivo é criado, gravado ou apagado. Voltaremos a esse assunto mais adiante.

> **MICRO DICAS**
>
> **COMO LIGAR E DESLIGAR O ACIONADOR**
>
> Ao ligar ou desligar o acionador, verifique se não há algum disco em seu interior. Como os disquetes são muito sensíveis, um setor do catálogo interno pode se apagar quando o acionador for ligado ou desligado.
>
> Esse acidente ocorre com frequência nos modelos cujo catálogo se situa nas trilhas mais externas, onde usualmente a cabeça fica em repouso. Recuperar a informação contida no restante do disco será, então, impossível, pois o software operacional não terá meios de localizá-la.

COMO USAR O DISQUETE

O conhecimento dos comandos de software é fundamental para o uso correto de unidades de disquetes, assim como de outros periféricos, entre os quais joysticks, canetas ópticas e impressoras. Esses comandos podem fazer parte do sistema operacional, das linguagens de programação (como o BASIC) ou, ainda, de ambos.

Como a sintaxe e o uso dos comandos variam conforme o micro, a linguagem e o sistema operacional, trataremos de cada máquina separadamente.



Os comandos utilizados pelos micros compatíveis com o TRS-Color para gravar e ler programas são o **SAVE** e o **LOAD** — assemelham-se, portanto, aos que se empregam com fita cassete (**CSAVE** e **CLOAD**). A maneira de usá-los também é bastante simples: se você quiser, por exemplo, gravar em disco um programa que está em memória, digite o comando **SAVE "NOME"**. Para carregar, digite **LOAD "NOME"**. Como o TRS-Color não tem sistema operacional separado do BASIC, trabalhar com uma unidade de disquetes é mais fácil.

Para carregar um programa em BASIC armazenado em disco, não precisa digitar o sufixo: basta indicar o nome dentro de um comando **LOAD**.

Quando se liga o computador pela primeira vez, a opção **VERIFY** é acionada, ou seja, tudo o que for gravado em disco será automaticamente verificado pelo computador. Para desativar essa opção, digite **VERIFY OFF**. Para reativá-la, digite **VERIFY ON**.

A lista dos arquivos gravados em um disquete será obtida com o comando:

```
DIR
```

que é uma abreviação de **DIR**ectory. Esse comando é muito útil, pois mostra na tela quantos bytes ocupa cada arquivo, se se trata de programa ou de dados, quais as suas características etc. **DIR** indica ainda quantos bytes estão sobrando no disquete.

Existe também um comando que possibilita a troca do nome de um arquivo já armazenado em disco. Se você digitar, por exemplo:

```
RENAME "VELHO" TO "NOVO"
```

o comando substituirá o nome do arquivo chamado VELHO por NOVO.

Para apagar arquivos do disco, utilize o comando **KILL**. Por exemplo, para um programa chamado INUTIL, digite:

```
KILL "INUTIL"
```



Os comandos utilizados pelos micros compatíveis com o TRS-80 para gravar e ler programas são o **SAVE** e o **LOAD** — assemelham-se, portanto, aos que se empregam com fita cassete (**CSAVE** e **CLOAD**). A maneira de usá-los também é simples: se você quiser, por exemplo, gravar em disco um programa que está em memória, digite **SAVE "NOME"**. Para carregar, digite **LOAD "NOME"**. O TRS-80 tem um sistema operacional separado do BASIC, mas os comandos de disco mais importantes também estão disponíveis na linguagem.

Para carregar um programa em BASIC armazenado em disco, não é necessário digitar o sufixo: basta indicar o nome dentro de um comando **LOAD**.

Para obter uma lista dos arquivos gravados em um disquete, a partir do sistema operacional, digite:

```
DIR
```

que é uma abreviação de **DIR**ectory. Esse comando é muito útil, pois mostra na tela quantos bytes ocupa cada arquivo, se se trata de programa ou de dados, quais as suas características etc. **DIR** indica ainda quantos bytes estão sobrando no disquete. O comando equivalente em BASIC é o **FILES**, de efeito semelhante, mas não igual ao do **DIR**

Existe também um comando que possibilita a troca do nome de um arquivo já armazenado em disco. Digite, por exemplo, em BASIC:

```
RENAME "VELHO" TO "NOVO"
```

e o comando substituirá o nome do arquivo chamado VELHO por NOVO.

Para apagar arquivos do disco, utilize o comando **KILL**. Suponhamos que você queira apagar um programa chamado INUTIL. Digite, em BASIC:

```
KILL "INUTIL/BAS"
```



Os comandos empregados pelos micros compatíveis com as linhas Apple e TK-2000 para gravar e ler programas são o **SAVE** e o **LOAD**. A maneira de usá-los é muito simples: se você quiser, por exemplo, gravar em disco um programa que está em memória, digite **SAVE NOME**. Para carregar, digite **LOAD NOME**. O Apple e o TK-2000 não têm um sistema operacional separado do BASIC, sendo fáceis de usar.

Para obter uma lista dos arquivos gravados em um disquete, a partir do sistema operacional, digite:

```
CATALOG
```

Esse comando é muito útil, pois mostra na tela quantos bytes ocupa cada arquivo, se se trata de programa ou de dados etc.

Existe também um comando que possibilita a troca do nome de um arquivo já armazenado em disco. Digite, por exemplo, em BASIC:

```
RENAME VELHO, NOVO
```

e o comando substituirá o nome do arquivo chamado VELHO por NOVO.

Para apagar arquivos do disco, utilize o comando **DELETE**. Suponhamos que você queira apagar um programa chamado INUTIL. Digite, em BASIC:

```
DELETE INUTIL
```

Outro comando útil é o **VERIFY**, que informa se determinado arquivo foi danificado ou escrito incorretamente. Merecem ainda menção o comando **LOCK** e o **UNLOCK**, que permitem proteger ou retirar a proteção de arquivos contra apagamento acidental.



Os comandos utilizados pelos micros da linha MSX para gravar e ler programas são o **SAVE** e o **LOAD** — os mesmos, portanto, que se empregam com fita cassete. O modo de usá-los é simples: se você quiser, por exemplo, gravar em disco um programa que está em memória, digite **SAVE "A:NOME"**, para registrar um arquivo chamado NOME, na unidade de disquete A. Para carregar um programa, digite **LOAD "A:NOME"**. O MSX pode acomodar até dois disquetes, chamados de A e B.

Quando quiser carregar um programa em BASIC armazenado em disco, não digite o sufixo; basta indicar o nome dentro de um comando **LOAD**.

Para obter uma lista dos arquivos gravados em um disquete, a partir do sistema operacional, digite, em BASIC:

```
FILES
```

Esse comando é muito útil, pois mostra na tela quantos bytes ocupa cada arquivo, se se trata de programa ou de dados, quais as suas características etc.

Existe também um comando que possibilita a troca do nome de um arquivo já armazenado em disco. Digite, por exemplo, em BASIC:

```
NAME "A:VELHO" AS "A:NOVO"
```

e o comando substituirá o nome do arquivo chamado VELHO por NOVO (ambos no disquete A).

Para apagar arquivos do disco, utilize o comando **KILL**. Suponhamos que você queira apagar um programa chamado INUTIL. Digite, em BASIC:

```
KILL "A:INUTIL BAS"
```



Os comandos **SAVE** e **LOAD** utilizados para a transmissão de dados entre computador e fita cassete foram modificados para permitir a gravação de leitura de programas no Microdrive:

```
LOAD *"m";1;"nome"
SAVE *"m";1;"nome"
```

O asterisco indica que o comando faz parte do BASIC armazenado na ROM da Interface 1. O **m** e o **1** informam que a operação se dará no Microdrive nº 1. Seguindo o mesmo formato, há um comando que verifica gravações depois de feitas (**VERIFY**) e outro que combina dois programas na memória.

O comando **CAT** — abreviatura de **CAT**álogo — é utilizado para exibir na tela a lista de arquivos armazenados em um cartucho de Microdrive.

Para apagar um arquivo do Microdrive, emprega-se o comando **ERASE**, que tem o mesmo formato que o **SAVE** e o **LOAD**, não exigindo, porém, o asterisco.



CUIDADOS COM A UNIDADE DE DISCOS

Ao contrário da UCP, do teclado e do vídeo, que são bastante resistentes e praticamente dispensam quaisquer reparos, as unidades de disco constituem o "calcanhar-de-Aquiles" dos sistemas baseados em microcomputadores. A confiabilidade de operação dessas unidades é muitas vezes maior do que a dos gravadores cassete, por exemplo. Porém, por serem dispositivos mecânicos relativamente complexos e muito delicados, exigem alguns cuidados por parte do usuário, se ele quiser operar por muitas horas, sem problemas.

Devido à sua fragilidade mecânica, as unidades de disquete são extremamente sensíveis a qualquer choque. Mesmo que nada se quebre depois de uma batida mais forte, pode ocorrer um descalibramento da cabeça. Portanto, mantenha a unidade em um ponto seguro da mesa. Se precisar transportá-la, faça-o com todo cuidado e dentro da embalagem acolchoada original.

É importante, também, levar em conta que a cabeça da unidade de disquete trabalha em atrito direto contra o meio magnético. Qualquer poeira, ou detrito, por menor que seja, que tiver acesso a ela, diminuirá a vida da mesma e provocará defeitos.

A temperatura ambiente deve ser mantida em níveis razoáveis durante a operação, sobretudo porque, normalmente, a unidade não possui ventilação.

Quanto aos cuidados gerais, lembre-se de que o cabo de conexão deve estar bem inserido, sem dobras e livres de tensões mecânicas. Limpe periodicamente os contatos dos conectores. Preste muita atenção ao inserir o disquete, para não fazê-lo com a orientação errada, o que pode danificar irremediavelmente a cabeça da unidade.

Evite usar disquetes "flippy", pois soltam fragmentos. Não use produtos de limpeza internamente: para isso, existem disquetes especiais de limpeza, que devem ser rodados a cada cinqüenta a cem horas de uso.



Quais são as vantagens da utilização de cartuchos de programas, em vez de fitas e discos? Posso gravar meus próprios programas em um cartucho?

A maior vantagem dos programas em cartuchos é que eles estão imediatamente disponíveis para uso, não sendo preciso carregá-los. Além disso, são muito mais seguros e resistentes contra a perda de programas.

Os cartuchos (como os existentes para micros das linhas TRS-Color e MSX) são memórias ROM removíveis, que não podem ser alteradas de nenhuma maneira — em outras palavras, constituem um meio permanente de armazenamento. Você não pode, portanto, gravar seus programas neles. Mas há a alternativa de gravação permanente de programas em um tipo de memória chamado EPROM (*Eraseable Programmable Read Only Memory*), o que requer um periférico especial, o programador de EPROM, que pode ser conectado a muitos micros. O custo da montagem de um sistema desse tipo é, porém, muito elevado.

OS SEGREDOS DO TRS-80 (1)

- RECURSOS OCULTOS DO TRS-80
- A MEMÓRIA DO VÍDEO
- O USO DOS COMANDOS PEEK e POKE
- COMO COPIAR A TELA

A memória ROM do TRS-80 guarda recursos que, em geral, os usuários desconhecem. Sabendo quais são eles, você poderá utilizar incríveis truques de programação em BASIC.

Os microcomputadores da linha TRS-80 são considerados tecnicamente obsoletos por muitas pessoas. Entretanto, os usuários desse popular modelo, que já tiveram a oportunidade de explorar mais a fundo seus diversos recursos, podem testemunhar o quanto eles são poderosos — algumas vezes, mais até do que os disponíveis em certos microcomputadores de tipo profissional, maiores e bem mais caros.

Na série de artigos que aqui iniciamos, você verá como explorar alguns dos recursos menos conhecidos do TRS-80 e de seus compatíveis nacionais (como o CP-300 e o CP-500, da Prológica) e internacionais. Daremos exemplos de manipulação de tela, teclado, som e de vários aspectos do BASIC pouco esclarecidos no Manual de Programação.

TRUQUES DE VÍDEO

O vídeo do TRS-80 possui uma propriedade muito interessante: ele é *mapeado em memória*. Isso significa que uma parte da memória RAM é dedicada exclusivamente ao vídeo, e que qualquer coisa nela armazenada terá, automaticamente, um efeito sobre o vídeo, sobretudo se se tratar de um caractere visível (ASCII ou gráfico).

A memória de vídeo começa na locação 15360 e ocupa 1024 bytes, ou seja, um para cada posição na tela. Há, portanto, uma equivalência entre o número indicado em um **PRINT@** e a memória absoluta referente à posição na tela, que é a soma de 15360 a esse número. Por exemplo, o comando

```
PRINT @ 125,"A",
```

colocará o código 65 (ASCII para a letra A maiúscula) na locação 15360 + 125, ou 15485.

PEEK E POKE

Como a memória de vídeo tem uma correspondência byte a byte com a tela, podemos escrever utilizando o comando **POKE**. A linha

```
POKE 15485,65
```

tem exatamente o mesmo efeito que o **PRINT@** mostrado acima.

Se você não quiser recorrer à tabela de códigos ASCII sempre que for escrever na tela com o **POKE**, use esta forma alternativa:

```
POKE 15485,ASC("A")
```

Em alguns casos, o emprego do **POKE** é mais vantajoso que o do **PRINT@** Para colocar caracteres na tela sem provocar o movimento do cursor, por exemplo, o **POKE** é o comando indicado. Além disso, em certas situações ele pode ser mais rápido do que o **PRINT@**. Finalmente, o comando **POKE** permite a impressão de um caractere no canto inferior direito da tela (posição 1023), sem provocar o deslocamento de toda a tela para cima (*scrolling*).

Quando se trata, porém, de imprimir uma cadeia de caracteres (armazenados em uma variável literal, por exemplo), o uso do **POKE** fica em nítida inferioridade em relação ao do **PRINT**.

Para examinar o conteúdo de uma locação da tela, utiliza-se também a função **PEEK**, o que pode ser muito útil em uma grande variedade de programas, principalmente de jogos. O programa abaixo, por exemplo, lê o que está escrito na linha de cima da tela e, automaticamente, transforma em minúsculas todas as letras.

```
10 CLS
20 INPUT "ENTRE MENSAGEM ";I$
30 CLS:PRINT I$
40 FOR I=15360 TO 15423
50 J=PEEK(I) IF J>64 AND J<91
THEN POKE I,J+32
60 NEXT I:PRINT
```

O mesmo poderia ser feito tomando-se, um a um, os caracteres de **I$**, transformando-os em uma outra cadeia. Porém, o programa ficaria mais complicado e menos rápido. Note que a transformação de tipos maiúsculos para tipos minúsculos é bem fácil: basta, para isso, somar 32 ao código ASCII do caractere maiúsculo.

Um programa de jogo pode nos fornecer um exemplo menos óbvio de aplicação do **PEEK**. Suponhamos que seja necessário testar se uma bala de canhão (um caractere gráfico) colidiu com alguma parte da estrutura complexa de um castelo. Será bem mais conveniente utilizar um **PEEK** a cada posição de deslocamento da bala, uma vez que a função **POINT** não terá um desempenho satisfatório nesse caso.

UMA CÓPIA DA TELA

Muitos programas exigem que se faça uma cópia rápida da tela, em alguma parte da memória que não seja a de vídeo. Esse procedimento é empregado, por exemplo, nos programas que criam "janelas" de tela ou superpõem várias informações sobre o vídeo. Neste último caso, poderemos precisar de diversas cópias da tela, uma de cada "camada" superposta de informações, para que se torne possível a recuperação das telas originais posteriormente.

Uma maneira fácil, mas não rápida, de copiar uma tela consiste em usar os comandos **POKE** e **PEEK** dentro de um laço que percorre cada memória de vídeo. Para isso, é necessário reservar a parte da memória RAM que conterá a cópia. A reserva é feita logo que se aciona o interpretador BASIC, aparecendo na tela a pergunta **MEM?**, **MEMORY SIZE?** ou outra, conforme a versão do computador utilizado.

O programa abaixo usa a memória RAM acima de 30000:

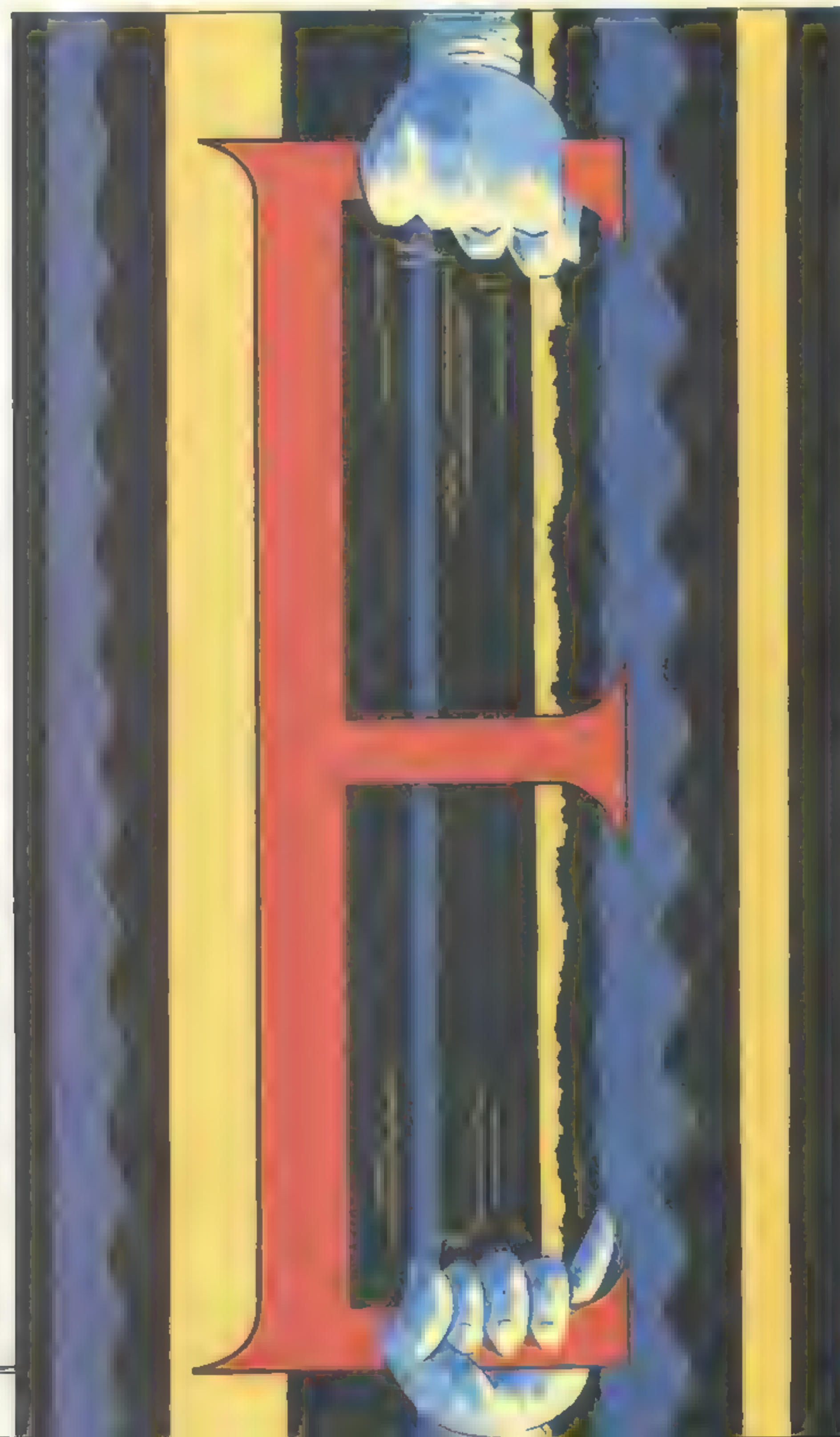
```
100 IM=30001
110 FOR I=15360 TO 16384
120 POKE IM,PEEK(I)
130 IM=IM+1:NEXT
```

Para trazer a cópia de volta para o vídeo, basta trocar os endereços do **PEEK** e do **POKE**:

```
100 IM=30001
110 FOR I=15360 TO 16384
120 POKE I,PEEK(IM)
130 IM=IM+1 NEXT
```

MANCHETES E LETREIROS (1)

- AMPLIAÇÃO DOS CARACTERES DA ROM
- ALTURA DAS LETRAS
- MONTAGEM DE LETRAS COM BLOCOS GRÁFICOS



Se você quiser escrever uma manchete ou um título em letras garrafais, precisará criar tipos especiais. Veja aqui dois diferentes métodos para a obtenção de caracteres ampliados.

O conjunto de caracteres disponíveis através do teclado deixa muito a desejar — o que é compreensível se considerarmos que ele foi criado de maneira a economizar espaço de memória. Assim, quando queremos escrever com letras maiores, destacando a mensagem, precisamos criar nossos próprios caracteres ampliados.

Existem muitas formas de utilizar os recursos gráficos de seu microcomputador para obter novos conjuntos de caracteres. A escolha de uma delas dependerá do tipo de aplicação que você tem em vista. Para ampliações, dispomos de duas alternativas básicas: desenhar as letras linha por linha no modo gráfico de alta resolução ou montar as letras com blocos gráficos.

Ambos os métodos podem ser usados dentro de um programa específico, destinado a escrever determinada frase — mas este é um caminho pouco econômico, já que cada mensagem exige uma nova rotina. A melhor solução, sobretudo quando pretendemos utilizar manchetes com freqüência, consiste em criar um programa que se encarregue de gerar os caracteres necessários. Uma vez feito isso, poderemos simplesmente informar ao computador o texto que deverá ser impresso, deixando a execução do serviço por conta do programa. Nosso trabalho se resumirá, assim, à gravação do gerador de letras e à sua colocação nos programas que precisem de caracteres maiores no vídeo.

Este é o primeiro de dois artigos em que são explicadas as diversas maneiras de escrever manchetes e cartazes. Aqui, trataremos em detalhe dos dois métodos já mencionados. O primeiro utiliza a tabela de padrões de caracteres existente na memória, aumentando seu formato. Como resultado, obtemos tipos semelhantes aos caracteres normais, só que em tamanho maior, disponíveis através

do teclado, da forma usual. Infelizmente, esse método não pode ser empregado pelos usuários do Apple e do TRS-Color, que não têm acesso aos padrões de caracteres.

O segundo método usa blocos gráficos da ROM para construir as letras, funcionando em todos os microcomputadores, menos no Apple, que não dispõe de blocos gráficos da ROM.

No próximo artigo, examinaremos outra maneira de criar letras, que pode ser adaptada a todo tipo de aplicação. Veremos, também, de que modo utilizar os métodos explicados no aperfeiçoamento de programas.

AMPLIAÇÃO DOS CARACTERES

Quando executamos o programa elaborado especialmente para ampliar caracteres, o computador aguarda a entrada da frase que deverá escrever. O Spectrum e o MSX, em seguida, colocam na tela as letras ampliadas, com o dobro da altura original.

Até adquirir alguma experiência e perceber como o programa funciona, escreva palavras curtas. Se usar frases mais longas, elas serão divididas no final da linha.

Com base nos bytes que definem os padrões dos caracteres, o programa cria os blocos gráficos que, juntos, formarão o novo caractere. Para duplicar a altura de uma letra, por exemplo, o computador substitui cada byte do caractere por dois outros, repetidos dentro do mesmo bloco gráfico.

S

```
1000 INPUT "INTRODUZA UM TEXTO"
, LINE I$: CLS : GOSUB 9000: GO
TO 1000
9000 LET line=0: LET col=-1
9010 LET y=0
9020 FOR i=1 TO LEN I$: LET col
=col+1: IF col=32 THEN  LET col
=0: LET line=line+2
9030 LET t$=I$(i)
9050 FOR x=0 TO 6 STEP 2
9060 POKE USR "a"+x,PEEK (15616
+(8*(CODE t$ 32))+y)
9070 POKE USR "a"+1+x,PEEK (156
16+(8*(CODE t$ 32))+y)
9080 LET y=y+1
9090 NEXT x
9100 FOR x=1 TO 7 STEP 2
9110 POKE USR "b"+x 1,PEEK (156
16+(8*(CODE t$-32))+y)
9120 POKE USR "b"+x,PEEK (15616
+(8*(CODE t$ 32))+y)
9130 LET y=y+1
9140 NEXT x
9150 LET y=0
9160 PRINT AT line,col;CHR$ 144
;AT line+1,col;CHR$ 145
9180 NEXT i
9200 RETURN
```

A primeira linha do programa possibilita a entrada de nosso texto, colocando-o dentro da variável alfanumérica **I$**. Para controlar a posição de impressão do texto na tela, utilizam-se duas variáveis. Seus valores iniciais são 0, para a coordenada vertical, e −1, para a coordenada horizontal. Uma terceira variável, **Y**, recebe o valor inicial zero, antes que o processo de ampliação comece.

A linha 9020 inicia um laço **FOR...NEXT** que dará um número de voltas igual ao comprimento de nossa frase. A cada volta, o valor da coordenada horizontal (variável **col**) aumenta em uma unidade. O valor inicial de **col** era −1 justamente para que a impressão do texto começasse a partir do canto esquerdo da linha, onde **col** = 0.

Se chegar a 32 (uma posição além da extremidade direita da linha), o valor da coordenada horizontal voltará a ser 0, para que o computador coloque as letras restantes a partir do lado esquerdo da tela. A coordenada vertical é, então, aumentada em duas unidades, tornando possível que o texto excedente seja escrito na linha seguinte.

A linha 9030 coloca a letra que deve ser ampliada dentro da variável **t$**, usando o contador do laço, **i** (**t$** conterá, assim, o i-ésimo caractere do texto). Em seguida, o computador dá início a um novo laço, que coloca dois bytes repetidos dentro de um bloco gráfico, quatro vezes.

A CHAVE DO PROCESSO

As contas feitas para calcular o número a ser colocado dentro do bloco constituem a chave de todo o processo de ampliação das letras.

Analisando a linha 9060, você poderá entender os cálculos. A primeira metade é muito simples. Ela coloca um número (cujo cálculo é explicado adiante) com **POKE** em um endereço **x** posições além do primeiro byte do bloco gráfico (UDG) **a**. A variável **x**, que controla o laço **FOR...NEXT**, aumenta com um **STEP 2**, de forma que o segundo byte não sofre alterações quando o laço é executado novamente.

A segunda metade calcula o número que será colocado pelo **POKE**. Esse número corresponde a um dos bytes do padrão do caractere que está sendo amplia-



do, obtido da tabela que fica na ROM. O endereço inicial da tabela é 15616. A fórmula entre parênteses subtrai 32 do código ASCII do caractere e multiplica o resultado por 8, para identificar a posição do primeiro byte do padrão dentro da tabela.

O código ASCII do caractere precisa ser subtraído em 32 unidades, antes da multiplicação por 8, devido ao modo como o Spectrum armazena os padrões dos caracteres (de fato, não há nessa parte da memória oito bytes para cada um dos 32 primeiros caracteres).

Depois de calcular a posição dos padrões do caractere dentro da tabela, o computador soma o valor obtido ao endereço inicial da tabela, 15616, e adiciona o resultado a y. Lembre-se de que no início do programa atribuímos a y o valor zero. Essa variável será utilizada para indicar o byte que está sendo lido e colocado dentro do UDG. Quando seu valor é zero, o computador lê e coloca no UDG o primeiro byte do caractere.

As linhas 9060 e 9070 colocam em duas linhas consecutivas do UDG o mesmo byte obtido na tabela de padrões, duplicando a altura da letra. Depois, **y** é aumentado em uma unidade e o processo se repete para o próximo byte do padrão, até que o laço **FOR... NEXT** termine — a esta altura, quatro pares de bytes terão sido colocados no UDG **a**.

O Spectrum inicia, então, outro laço **FOR...NEXT**, que procede como o anterior para colocar os quatro últimos bytes do padrão do caractere no UDG **b**, sempre repetindo cada byte duas vezes. O laço termina na linha 9140 e o programa faz y voltar a valer 0.

A linha 9160 é a responsável pela impressão dos blocos gráficos, um acima do outro. Ela usa as variáveis **line** e **col** para identificar a posição de impressão do caractere ampliado. No programa, o comando **PRINT "A"** foi substituído por **CHR$ 144** para evitar que o leitor se confunda — o "A" normal e o "A" gráfico pareceriam absolutamente iguais na listagem.

Finalmente, a linha 9180 conclui o laço principal do programa, mandando o computador ampliar a letra seguinte. Ao completar a frase, o computador volta à linha 1000, que possibilita a entrada de um novo texto.



Embora o Apple não nos dê acesso aos padrões binários que definem o formato de seus caracteres, veremos como aplicar a técnica de ampliação utilizando os UDG criados por um programa do artigo da página 526.

Na listagem que se segue, você encontrará apenas as modificações necessárias para ampliar as letras criadas no programa mencionado. Sem as linhas **DATA** iniciais, que se encontram listadas naquele artigo, você não obterá nenhum resultado. Mais ainda, se tentar executar apenas as linhas aqui apresentadas, o computador sairá fora de seu controle e as linhas digitadas acabarão se perdendo.

```
799  HGR
800 A$ = "ESTA MENSAGEM E' UM T
ESTE"
810 C = 7·L = 11· GOSUB 1000
820  END
1000 N = L * 40 + C
1010  FOR J = 1 TO  LEN (A$)
1020 B$ =  MID$ (A$,J,1)
1030 B =  ASC (B$)
1040 L =  INT (N / 40):C = N -
40 * L
1050  FOR I = 0 TO 3
1060  POKE T + (L - 8 * (L > 7)
 - 8 * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * 2 * I, PEEK (E + B * 8 + I
),
1065  POKE T + (L - 8 * (L > 7)
```

```
 - 8 * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * (2 * I + 1), PEEK (E + B *
 8 + I)
1066  NEXT I:L = L + 1
1070  FOR I = 0 TO 3
1073  POKE T + (L - 8 * (L > 7)
 - 8 * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * 2 * I, PEEK (E + B * 8 + I
 + 4)
1076  POKE T + (L - 8 * (L > 7)
 - 8 * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * (2 * I + 1), PEEK (E + B *
 8 + I + 4)
1079  NEXT I:N = N + 1: NEXT J
1080  HCOLOR= 6
1090  HPLOT 25,80 TO 250,80 TO
250,105 TO 26,105 TO 26,80
1100  RETURN
```

Consultando o artigo anteriormente citado, veremos que nosso programa utiliza a sub-rotina que começa na linha 1000 para colocar os padrões das letras da frase contida em **A$** na tela de alta resolução. As variáveis **C** e **L** definem a linha e a coluna em que faremos a impressão, sempre supondo que a tela tem quarenta colunas e 24 linhas.

A linha 1000 coloca em **N** a posição da tela em que a primeira letra da frase será escrita. A linha 1010 inicia um laço **FOR...NEXT** que dará tantas voltas quantas forem as letras da frase que desejamos imprimir.

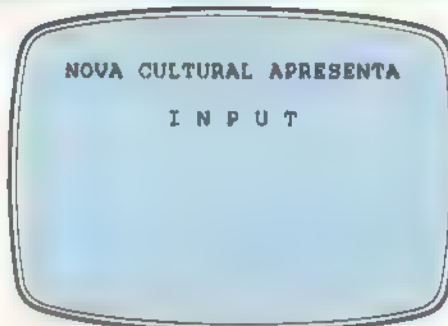
A variável alfanumérica **B$** contém a letra que está sendo escrita. A cada volta, um novo caractere é colocado ali, com o auxílio da função **MID$**, e a linha e a coluna de impressão são recalculadas a partir do valor de **N**.

Em seguida, o programa inicia um laço **FOR...NEXT** que coloca na tela dois bytes repetidos do padrão da letra, quatro vezes. Os cálculos feitos para a repetição dos bytes são a chave do processo de ampliação.

FUNCIONAMENTO DO PROGRAMA

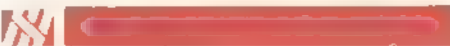
Já tivemos a oportunidade de mostrar aos usuários do Apple e do TK-2000 como é caótica a organização da memória de vídeo desses dois micros. A ordem de preenchimento das linhas de pixels do Apple é tão confusa que quem não tem certa familiaridade com a matemática dificilmente conseguirá entender as fórmulas das linhas 1060 a 1076.

Mas, no nosso caso, isso não tem tanta importância. Para compreender o funcionamento do programa, basta saber que o primeiro laço coloca na tela a metade superior da letra ampliada e o segundo, a metade inferior. Um caractere comum ocuparia oito linhas de vídeo com seus oito bytes. Aqui, para ampliar as letras, cada byte é repetido duas vezes, resultando nas dezesseis linhas do caractere ampliado.



Letras de altura dupla são ideais para páginas-título de jogos ou outros programas.

Assim, a linha 1060 coloca os quatro primeiros bytes do padrão da letra nas posições pares do bloco gráfico correspondente à metade superior do caractere ampliado. A linha 1065, por sua vez, coloca os mesmos quatro primeiros bytes nas posições ímpares daquele bloco. O processo se repete nas linhas 1073 e 1076, só que os bytes são colocados no bloco correspondente à metade inferior do caractere.



```
10 FOR J=0 TO 50 STEP 2
20 FOR X=0 TO 7 STEP 2
30 VPOKE BASE(2)+1024+J*8+X,PEE
K((PEEK(4))+(PEEK(5)*256)+520+Y
)
40 VPOKE BASE(2)+1024+J*8+X+1,P
EEK((PEEK(4))+(PEEK(5)*256)+520
+Y)
50 LET Y=Y+1
60 NEXT X
70 FOR X=8 TO 15 STEP 2
80 VPOKE BASE(2)+1024+J*8+X,PEE
K((PEEK(4))+(PEEK(5)*256)+520+Y
)
90 VPOKE BASE(2)+1024+J*8+X+1,P
EEK((PEEK(4))+(PEEK(5)*256)+520
+Y)
100 LET Y Y+1
110 NEXT X
120 NEXT J
130 CLS
140 PRINT"INTRODUZA A PALAVRA"
150 INPUT I$
160 CLS L=0:C=0
170 FOR I=1 TO LEN(I$)
180 LET B ASC(MID$(I$,I,1))
190 IFB<65 OR B>90 THEN GOTO 21
0
200 LOCATE C,L PRINTCHR$(128+2*
(B 65)):LOCATE C,L+1:PRINTCHR$(
129+2*(B-65))
210 LET C=C+1
220 IF C=40 THEN LET C 0 L=L+2
230 NEXT I
240 GOTO 140
```

O programa começa com a rotina que transfere os dados de definição dos caracteres da ROM para a memória de vídeo (VRAM), nas linhas 10 a 120.

A seção que vai da linha 130 à 150 limpa a tela e pede que o usuário introduza a palavra ou mensagem a ser ampliada. As linhas 160 a 240 contêm a rotina que imprime as letras no vídeo. As variáveis **L** e **C** registram, respectivamente, a linha e a coluna em que serão impressos os novos caracteres gráficos que irão compor as letras.

O laço que se inicia na linha 170 imprime uma letra a cada valor de **I**, sempre acrescentando 1 à variável **C**, para que as letras fiquem uma ao lado da outra. Quando a frase alcança a extremidade da tela, a linha 200 zera **C** e acrescenta 2 à variável **L**, a fim de que a próxima letra seja escrita na primeira coluna da linha seguinte.

A linha 180 faz a variável **B** assumir o valor do código ASCII do caractere a ser ampliado e a linha 190 verifica se as letras são maiúsculas. A linha 200 imprime os dois blocos gráficos que irão compor a letra, ambos na mesma coluna, mas um abaixo do outro.

Para saber como se obtêm os valores colocados dentro dos parênteses após o comando **CHR$**, é preciso entender o processo de ampliação.

O PROCESSO DE AMPLIAÇÃO

Ao ser ligado, o MSX copia o padrão dos caracteres gravados na ROM na parte de sua memória dedicada ao vídeo (VRAM). É a partir dessa tabela da VRAM que ele imprime as letras no vídeo. Cabe ao nosso programa criar os blocos gráficos que irão compor as letras ampliadas e colocá-los na tabela da VRAM, no lugar que antes era ocupado pelos caracteres padronizados, a partir do caractere de código 128.

Sabemos que o padrão de um caractere é formado por oito bytes. Podemos, portanto, obter um caractere ampliado repetindo duas vezes cada byte. O endereço do padrão dos caracteres na ROM está registrado nos bytes 4 e 5 e o endereço do padrão de caracteres na VRAM é dado por **BASE(2)**. Somando o endereço aí contido a 1024, teremos o endereço da VRAM, que será usado pelo **VPOKE**.

Na linha 30, a viriável **J** simplesmente incrementa a posição nessa memória. O **PEEK** examina o endereço correspondente na RAM, onde estão os padrões gráficos. Para a obtenção do código da

letra A, adiciona-se o número 520. O Y indica o byte a ser copiado. As linhas 70 a 110 do programa fazem a mesma cópia para a parte inferior do bloco gráfico de cada caractere.

O USO DOS CARACTERES GRÁFICOS

Além de ampliar os padrões preexistentes na memória do micro, podemos criar nossas próprias letras. Essa alternativa é especialmente importante para os usuários do TRS-Color, que não permite o acesso aos padrões das letras. Aqui estão alguns programas que utilizam caracteres gráficos do computador para construir letras ampliadas.

S

```
10 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
20 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
30 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
40 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
50 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
60 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
70 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
80 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
90 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
100 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
110 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
120 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
130 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
140 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
150 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
160 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
170 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
180 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
190 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
200 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
210 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
220 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
230 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
240 DATA "[illegible]","[illegible]",
 "[illegible]","[illegible]"
250 DATA "[illegible]","[illegible]"
260 DATA "[illegible]","[illegible]"
270 DATA "[illegible]","[illegible]"
280 DATA "[illegible]","[illegible]"
290 POKE 23658,8: DIM a$(27,3)
 : DIM b$(27,3): DIM c$(27,3):
DIM d$(27,3)
300 FOR j=0 TO 21 STEP 4
310 FOR i=1 TO 4: READ a$(i+j)
 : NEXT i
320 FOR i=1 TO 4: READ b$(i+j)
 : NEXT i
330 FOR i=1 TO 4: READ c$(i+j)
 : NEXT i
340 FOR i=1 TO 4: READ d$(i+j)
 : NEXT i
350 NEXT j
360 FOR i=25 TO 26: READ a$(i)
 : NEXT i
370 FOR i=25 TO 26: READ b$(i)
 : NEXT i
380 FOR i=25 TO 26: READ c$(i)
 : NEXT i
390 FOR i=25 TO 26: READ d$(i)
 : NEXT i
400 INPUT "Introduza palavra(m
ax 10letras)", LINE t$: IF LEN
t$>10 THEN  LET t$=t$( TO 10)
405 IF LEN t$=0 THEN  GOTO 400
410 LET s$="": FOR i=1 TO LEN
t$: IF CODE t$(i)<65 OR CODE t
$(i)>90 THEN  LET t$(i)=CHR$
91
420 LET s$=s$+a$(CODE t$(i)-64
): NEXT i
430 PRINT s$
440 LET s$="": FOR i=1 TO LEN
t$
450 LET s$=s$+b$(CODE t$(i)-64
): NEXT i
460 PRINT s$
470 LET s$="": FOR i=1 TO LEN
t$
480 LET s$=s$+c$(CODE t$(i)-64
): NEXT i
490 PRINT s$
500 LET s$="": FOR i=1 TO LEN
t$
510 LET s$=s$+d$(CODE t$(i)-64
): NEXT i
520 PRINT s$
530 PRINT : GOTO 400
```

MSX

```
10 SCREEN 0
20 DIM A(78),B(78),C(78),D(78)
30 FOR I=1 TO 78:READ S$:A(I)=V
AL("&H"+S$):NEXT
40 FOR I=1 TO 78:READ S$:B(I)=V
AL("&H"+S$):NEXT
50 FOR I=1 TO 78:READ S$:C(I)=V
AL("&H"+S$):NEXT
60 FOR I=1 TO 78:READ S$:D(I)=V
AL("&H"+S$):NEXT
70 PRINT"introduza até 9 letras
"
80 INPUT T$:IF LEN (T$)>9 OR LE
N (T$)=0 THEN 80
90 FOR X=1 TO 4
100 FOR Y=1 TO LEN(T$)
110 A$=MID$(T$,Y,1)
120 A=ASC(A$)
```

MICRO DICAS

COMO USAR A IMPRESSORA PARA CONFECCIONAR CARTAZES E FAIXAS

Os programas elaborados para desenhar letras grandes na tela do microcomputador têm uma utilidade adicional: a impressão de grandes cartazes e faixas.

Como sabemos, as letras normais, obtidas em uma impressora comum para microcomputador, são muito pequenas para serem vistas a distância. Precisamos, assim, lançar mão de métodos semelhantes aos apresentados neste artigo para imprimir letras de tamanho maior.

A escolha da melhor solução dependerá, evidentemente, do tipo de impressora que você possui.

O tipo mais simples é aquele capaz de imprimir apenas texto, não dispondo de blocos gráficos. Nesse caso, programas que trabalham com blocos gráficos da ROM, como os destinados ao Spectrum, não podem ser utilizados diretamente com a impressora, pois ela não os copiará.

Uma boa alternativa será modificar os programas, de modo a compor letras grandes a partir dos caracteres disponíveis na impressora. Experimente, por exemplo, o recurso muito comum de usar repetições do próprio caractere, como mostramos, abaixo, com a letra I maiúscula

```
IIIIII
  II
  II
  II
  II
IIIIII
```

As linhas **DATA**, que contêm os códigos dos blocos gráficos, podem ser alteradas para descrever a disposição dos caracteres usados na composição. Outra providência necessária é mudar os comandos **PRINT** (que colocam o resultado apenas na tela) para o equivalente ao comando de impressão existente em seu computador: **LPRINT** (nos micros TRS-80, Sinclair e MSX), **PR # 1** (no Apple) e **PRINT # 1** (no TRS Color).

Se sua impressora tiver capacidade gráfica, o trabalho será ainda mais fácil. Usando um comando específico, como o **COPY**, nos micros da linha Sinclair, ou um programa de descarregamento de tela, você poderá transferir para o papel o cartaz desenhado na tela do computador com os programas aqui apresentados.

```
127 IF A<65 OR A>90 THEN CLS:GO
TO 70
129 LET A=A-64
130 B=(A-1)*3
140 ON X GOSUB 190,200,210,220
150 NEXT Y
160 PRINT
170 NEXT X
180 GOTO 70
190 FOR R=B+1 TO B+3·PRINTCHR$(
A(R));·NEXT R: PRINT" ";·RETURN
200 FOR R=B+1 TO B+3·PRINTCHR$(
B(R));·NEXT R·PRINT" ";·RETURN
210 FOR R=B+1 TO B+3:PRINTCHR$(
C(R))::NEXT R:PRINT" ";·RETURN
220 FOR R=B+1 TO B+3:PRINTCHR$(
D(R));·NEXT R·PRINT" ";:RETURN
230 DATA C7,DF,D6,DB,DF,D6
240 DATA C7,DF,D6,DB,DF,D6
250 DATA DB,DF,D3,DB,DF,D3
260 DATA DB,DF,DD,DB,20,DD
270 DATA D5,DB,D3,D5,DB,D3
280 DATA DB,20,DD,DB,20,20
290 DATA DB,DE,DD,DB,20,DD
300 DATA C7,DF,D6,DB,DF,D6
310 DATA C7,DF,D6,DB,DF,D6
320 DATA C7,DF,D6,DF,DB,D3
330 DATA DB,20,DD,DD,20,DD
340 DATA DD,20,DD,DD,20,DD
350 DATA DD,20,DD,DF,DF,D6
360 DATA DD,20,DD,DB,DC,D3
370 DATA DD,20,20,DB,20,DD
380 DATA DB,DC,20,DB,DC,20
390 DATA DB,20,20,DB,DC,DD
400 DATA 20,DB,20,20,DB,20
410 DATA DB,C7,20,DB,20,20
420 DATA DD,DD,DD,DD,DD,DD
430 DATA DD,20,DD,DB,20,DD
440 DATA DD,20,DD,DB,20,DD
450 DATA C1,DC,20,20,DB,20
460 DATA DB,20,DD,DD,20,DD
470 DATA DD,D6,DD,D5,C7,20
480 DATA C1,D4,D3,20,C7,20
490 DATA C7,DF,D6,DB,20,DD
500 DATA DD,20,20,DB,20,DD
510 DATA DB,20,20,DB,20,20
520 DATA DB,D5,DD,DB,20,DD
530 DATA 20,DB,20,D6,DB,20
540 DATA DB,C1,20,DB,20,20
550 DATA DD,20,DD,DD,DD,DD
560 DATA DD,20,DD,DB,DF,20
570 DATA DD,D6,DD,DB,DF,D6
580 DATA 20,20,DD,20,DB,20
590 DATA DB,20,DD,DE,DE,20
600 DATA DD,DD,DD,D4,C1,20
610 DATA 20,DD,20,D4,D3,20
620 DATA DD,20,DD,DB,DC,D3
630 DATA C1,DC,D3,DB,DC,D3
640 DATA DB,DC,D6,DB,20,20
650 DATA DB,DC,DD,DB,20,DD
660 DATA D4,DB,D6,C1,DB,20
670 DATA DB,20,DD,DB,DC,DD
680 DATA DD,20,DD,DD,DE,DD
```

```
690 DATA C1.DC.D3.DB,20,20
700 DATA C1.C7.D6,DB,20.DD
710 DATA C1,DC,D3,20.DB.20
720 DATA D8,DC.DD.20.DD,20
730 DATA C1,C1,D3.DD.20.DD
740 DATA 20,DD,20.C1,DC.D6
```

T

```
10 DIM L(2,3,25)
20 FOR J=0 TO 25:FOR K=0 TO 3·F
OR L=0 TO 2:READ L(L,K,J):NEXT
L,K,J
30 CLS0
40 PRINT @480,"INTRODUZA A PALA
VRA -";B$;
50 A$=INKEY$:IF(A$<"A" OR A$>"Z
")AND A$<>CHR$(8) AND A$<>CHR$(
13) AND A$<>" " THEN 50
60 IF A$=CHR$(13) THEN 110
70 IF A$=CHR$(8) AND B$="" THEN
 50
80 IF A$=CHR$(8) THEN B$=LEFT$(
B$,LEN(B$)-1):GOTO 30
90 IF LEN(B$)>9 THEN 50
100 B$=B$+A$.GOTO 30
110 IF B$="" THEN CLS:END
120 CLS0:PRINT @480,"COR (1-8)
?";
130 A$=INKEY$:IF A$<"1" OR A$>"
8" THEN 130
140 CLS0:CL=VAL(A$)
150 FOR Y=0 TO 3:FOR C=1 TO LEN
(B$):FOR X=0 TO 2
160 IF MID$(B$,C,1)=" " THEN PR
INT CHR$(128);:GOTO 180
170 PRINT CHR$(84+CL*16+L(X,Y,A
SC(MID$(B$.C,1))-65));
180 NEXT X.C:PRINT STRING$(32-P
OS(0),128);:NEXT Y
190 B$="":GOTO 40
1000 DATA 42,40,38,38,28,38,42,
40,38,38,28,38,42,40,30,39,31,3
6,38,28,38,39,31,36
1010 DATA 42,40,38,38,28,28,38,
28,28,39,31,38,42,40.30.38.28.3
8.38,28,38,39,31,36
1020 DATA 42,40,36,39.31,30.38.
28,28,39,31,30,42,40,36,38.28,2
8,42,40,36,38,28,28
1030 DATA 42,40,38,38,28,28,38,
29,30,39,31,38,38,28,38,39,31,3
8.38,28,38,38,28,38
1040 DATA 40,42,36,28,38,28,28.
38,28.31,39,30.28.41,36,28,33,2
8.28,33,28,39,35,28
1050 DATA 38,29.36,39,36,28,42,
30,28,38,32,30,38,28,28,38,28,2
8,38,28.28,39,31,30
1060 DATA 39.29,38,38,38,38.38,
28,38,38,28,38,39,28,38,42,30.3
8,38,37,38,38,32,38
```



Letras maiúsculas no TRS-Color...



...e no Spectrum.

```
1070 DATA 42,40,38,38,28,38,38,
28,38,39,31,38,42,40,38,38,28,3
8,42,40,36,38,28,28
1080 DATA 42,40,38,38,28,38,38,
30,38,39,35,38,42,40,38,38,28,3
8,42,41,36,38,28,38
1090 DATA 42,40,38,39,31,30,28,
28,38,39,31,38,40,42,36,28,38,2
8,28,38,28,28,38,28
1100 DATA 38,28,38,38,28,38,38,
28,38,39,31,38,38,28,38,38,28,3
8,38,28,38,32,34,28
1110 DATA 38,28,38,38,28,38,38,
38,38,39,39,38,38,28,38,32,34,2
8,29,37,28,38,28,38
1120 DATA 38,28,38,39,31,38,28,
38,28,28,38,28,40,40,38,28,29,3
6,29,36,28,39,31,30
```

Os programas apresentados funcionam de maneira bem parecida. Eles começam dimensionando as matrizes que irão conter os caracteres gráficos da ROM necessários à formação das diversas letras. O Spectrum emprega o comando **POKE** para travar as letras maiúsculas. O Spectrum e o MSX usam quatro matrizes, uma para cada linha das letras. Os demais trabalham só com uma matriz.

Ao ser rodado, o programa demora um pouco para funcionar, devido à leitura das linhas **DATA** pelo comando **READ**. No Spectrum, essa tarefa é feita pelas linhas 290 a 390 do programa; no MSX, pelas linhas 30 a 60; nos demais computadores, só pela linha 20.

Em seguida, inicia-se a rotina principal, que possibilita ao usuário escrever uma palavra. Esta deve ter, no máximo, dez letras.

Depois de verificar se a cadeia de caracteres digitada é válida, o programa começa um laço **FOR...NEXT** que imprime uma das letras a cada volta. Esse laço é igual ao do último programa deste artigo, e só termina quando a última letra tiver sido impressa.

O Spectrum usa quatro laços **FOR...NEXT**, um para cada linha das letras, já que elas têm quatro caracteres de altura. O primeiro laço adiciona um grupo de três caracteres a **s$**, para cada uma das letras da frase. Quando não há mais letras, o computador imprime **s$**. O grupo de caracteres adicionado a **s$** varia conforme a linha impressa. Como podemos observar nas linhas 420, 450, 480 e 510, adiciona-se primeiro o cordão **a$**, depois o **b$**, o **c$**, e, finalmente, o **d$**. Cada um desses cordões é, de fato, uma matriz. Os números entre parênteses determinam qual elemento será adicionado a **s$**.



Assim que digitamos o texto e teclamos **ENTER**, o computador salta para a linha 110, que nos permite escolher a cor em que o texto será impresso. As linhas 130 e 140 se encarregam de acertar as cores e o programa prossegue com três laços **FOR...NEXT**.

A linha 160 imprime um bloco negro sempre que o caractere do texto for um espaço. Depois, o computador pula a linha 170, indo direto à 180.

A linha 170, embora pareça extremamente complicada, apenas imprime o próximo caractere gráfico da letra que está sendo desenhada. Os cálculos entre parênteses fornecem o código do caractere adequado. A primeira operação consiste na soma do número 84 ao código da cor multiplicado por 16. Segue-se a adição do resultado ao número apropriado da matriz. **L(X,Y,ASC(MID$(B$,C,1)) −65)** determina qual elemento da matriz **L** será usado nos cálculos.

Os números das linhas **DATA** — colocados depois na matriz **L** — correspondem aos códigos dos diversos caracteres gráficos (básicos, pretos e verdes) menos 100. Eles são distribuídos em grupos de doze (quatro linhas de três caracteres) para cada letra, de modo que o 13º número da lista corresponde à letra B, o 25º, ao primeiro caractere da letra C e assim por diante.

Quando o computador identifica o elemento da matriz que deve imprimir, a linha 170 usa a função **ASC** para calcular seu código e a função **MID$** para saber de qual letra da frase se trata. Os três números entre parênteses do **MID$** definem o cordão que será cortado (**B$**,em nosso caso), seu comprimento (**C**, em nosso caso) e o número de caracteres que se deve obter a partir deste ponto (1, em nosso caso).

O ponto e vírgula após a fórmula da linha 170 evita o salto de linha, fazendo com que as letras ampliadas sejam escritas uma ao lado da outra.

Na linha 180, duas instruções **NEXT** chamam os valores seguintes de **X** (que controla a impressão do caractere adequado da linha da letra) e de **C**.

A segunda parte da linha 180 imprime um número de blocos negros suficiente para que o computador passe à próxima linha e comece a montar a cadeia seguinte de caracteres gráficos — são quatro cadeias ao todo. Ao se completar o último laço, sua frase estará escrita na tela, na cor escolhida e em tamanho ampliado.

A linha 190 aguarda que outra tecla seja pressionada, para que o computador volte à linha 40 e permita a entrada de uma nova palavra.

Quando você quiser interromper o programa, voltando ao BASIC, bastará entrar uma frase "vazia" — ou seja, apertar **ENTER** antes de digitar qualquer letra — ou pressionar **BREAK**.



A introdução da palavra (com, no máximo, nove letras) é feita nas linhas 70 e 80. O laço que vai da linha 90 a 170 "desmonta" essa palavra, letra por letra, e desenha na tela cada uma das quatro camadas que formam um caractere, chamando as rotinas das linhas 190 a 220 através de um **ON × GOSUB** (linha 140). O código que será usado, calculado a partir do código ASCII do caractere, na linha 130 do programa, é armazenado em **B**.

As linhas 230 a 740 contêm os códigos gráficos que compõem as quatro camadas de cada caractere.



Os programas deste artigo mostraram duas maneiras de criar letras ampliadas. No próximo artigo, você verá como obter outro tipo de caractere e, também, como utilizar todos esses métodos em seus próprios programas.

MANCHETES E MAIS MANCHETES

- DESENHE LETRAS LINHA POR LINHA
- ALTURA E LARGURA
- DESENHE SEU PRÓPRIO TIPO DE LETRA

Além de blocos gráficos e do conjunto de caracteres do computador, existem outros recursos que podem ser utilizados na criação de letras. Este artigo mostra um deles.

Os dois tipos de letra criados pelo programa do artigo *Manchetes e Letreiros* (página 912) são construídos a partir de blocos gráficos ou do conjunto de caracteres-padrão da máquina. Mas há um outro meio de criar letras: usando gráficos de alta resolução para desenhá-las, linha por linha.

O programa que se segue funciona de modo semelhante ao dos blocos gráficos. Cada letra é definida por uma série de dados alocados em uma instrução **DATA**, que informam ao computador como ela será. Essa informação é armazenada em uma matriz, e é possível dar entrada às palavras que se quer escrever na forma de um cordão alfanumérico. Como antes, as letras são montadas conforme as instruções encontradas e exibidas na tela. Mais tarde, você aprenderá a usar essas telas como parte de seus próprios programas.

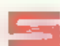


UM TIPO A SEU GOSTO

Os gráficos de alta resolução facilitam enormemente a tarefa de criar o seu tipo de letra preferido. Isso acontece porque a expansão dos caracteres da ROM é limitada à duplicação da altura ou da largura dos caracteres. Ora, as letras constituídas por blocos gráficos estão completamente fixadas devido aos números das linhas **DATA**.

Por outro lado, os dados no nosso programa também fixam a maneira como a letra vai ser desenhada (o L, por exemplo, é formado de uma linha vertical e uma horizontal). Mas essas instruções são relativas e não determinam a aparência final das letras. Pense em cada letra como se ela estivesse encerrada dentro de uma caixa imaginária. Se a caixa é alta e estreita, temos uma letra alta e estreita; se ela for baixa e larga, assim será a letra. O programa é escrito de tal forma que é possível definir essas variáveis com fatores de escala.

Agora digite o programa...



```
10 POKE 23658,8
20 DIM N(26): DIM A(26,12,2)
30 FOR N=1 TO 26
40 READ N(N)
50 FOR M 1 TO N(N)
60 READ A(N,M,1),A(N,M,2)
70 NEXT M
80 NEXT N
100 INPUT "INTRODUZA UMA PALAV
RA", LINE A$: IF A$="" THEN
GOTO 100
110 INPUT "DIGITE FATOR X",X
120 INPUT "DIGITE FATOR Y",Y
125 CLS
130 FOR N=1 TO LEN A$
.40 LET T$=A$(N)· IF T$<"A" OR
T$>"Z" THEN  NEXT N  GOTO 100
150 PLOT (.0*(N-1)*X)+X*A(CODE
T$-55,1,1),20+Y*A(CODE T$-55,1
,2)
160 FOR M=2 TO N(CODE T$ 55)
170 DRAW X*A(CODE T$-55,M,1),Y
*A(CODE T$ 55,M,2)
180 NEXT M
190 NEXT N
200 GOTO 100
1000 DATA 8,0,0,5,1,1,4,0,1, 1,
0,-5,0,3, 6,0
1010 DATA 12,0,0,0,6,5,0,1, 1,0
,-1,-1,-1,-5,0,5,0,1,-1,0,-1,-1
```

```
,-1,-5,0
1020 DATA 8,6,1,-1,-1,-4,0,-1,1
,0,4,1,1,4,0,1, 1
1030 DATA 7,0,0,0,6,4,0,2, 2,0,
-2,-2,-2,-4,0
1040 DATA 7,6,0,-6,0,0,6,6,0,-6
,0,0, 3,5,0
1050 DATA 6,0,0,0,6,6,0,-6,0,0,
-3,5,0
1060 DATA 10,5,2,1,0,0,-1,-1,-1
,-4,0,-1,1,0,4,1,1,4,0,1,-1
1070 DATA 6,0,0,0,6,0,-3,6,0,0,
3,0,-6
1080 DATA 6,0,0,6,0,-3,0,0,6,-3
,0,6,0
1090 DATA 5,0,1,1,-1,4,0,1,1,0,
5
1100 DATA 6,0,0,0,6,0,-3,6,3,-6
,-3,6, 3
1110 DATA 3,6,0, 6,0,0,6
1120 DATA 5,0,0,0,6,3,-3,3,3,0,
-6
1130 DATA 4,0,0,0,6,6,-6,0,6
1140 DATA 9,1,0,-1,1,0,4,1,1,4,
0,1,-1,0,-4, 1,-1,-4,0
1150 DATA 7,0,0,0,6,5,0,1,-1,0,
-1,-1,-1,-5,0
1160 DATA 9,4,2,2, 2,-5,0,-1,1,
0,4,1,1,4,0,1, 1,0,-5
1170 DATA 9,0,0,0,6,5,0,2,-1,0,
-1,-1,-1,-5,0,4,0,2,-3
1180 DATA 12,0,1,1,-1,4,0,1,1,0
,1,-1,1,-4,0,-1,1,0,1,1,1,4,0,1
,-1
1190 DATA 4,3,0,0,6, 3,0,6,0
1200 DATA 6,0,6,0, 5,1,-1,4,0,1
,1,0,5
1210 DATA 3,0,6,3, 6,3,6
1220 DATA 5,0,6,1, 6,2,3,2,-3,1
,6
1230 DATA 5,0,0,6,6,-3,-3,-3,3,
6,-6
1240 DATA 5,3,0,0,3,-3,3,3,-3,3
,3
1250 DATA 4,6,0,-6,0,6,6,-6,0
```

```
10 PMODE 1,1:PCLS
20 DIM N(26),A(26,12,2)
30 FOR N=1 TO 26
40 READ N(N)
50 FOR M=1 TO N(N)
60 READ A(N,M,1),A(N,M,2)
70 NEXT M,N
80 CLS INPUT"INTRODUZA A PALAVR
A ";A$:IF A$="" THEN GOTO 80
90 INPUT"QUAL O FATOR X ";X
100 INPUT"QUAL O FATOR Y ";Y
110 CLS PCLS SCREEN 1,0:FOR N=1
 TO LEN(A$)
120 T$=MID$(A$,N,1):IF T$<"A" O
R T$>"Z" THEN NEXT N:GOTO 80
130 J=(10*(N-1)*X)+X*A((ASC(T$)
-64),1,1):K=10+Y*A(ASC(T$)-64,1
,2)
140 FOR M=2 TO N(ASC(T$)-64)
150 F=X*A((ASC(T$)-64),M,1)
160 G=Y*A((ASC(T$)-64),M,2)
170 LINE(J,K)-(J+F,G+K),PSET
180 J=J+F K=K+G
190 NEXT M,N
200 IF INKEY$="" THEN 200
210 GOTO 80
1000 DATA 9,0,6,0,-5,1,-1,4,0,1
,1,0,3,0,2,0,-3,-6,0
1010 DATA 12,0,0,0,6,5,0,1,-1,0
,-1,-1,-1,-5,0,5,0,1,-1,0,-1,-1
,-1,-5,0
1020 DATA 8,6,1,-1,-1,-4,0,-1,1
,0,4,1,1,4,0,1,-1
1030 DATA 7,0,0,0,6,4,0,2,-2,0,
-2,-2,-2,-4,0
1040 DATA 7,6,0,-6,0,0,6,6,0,-6
,0,0,-3,5,0
1050 DATA 7,0,0,6,0,-6,0,0,3,5,
0,-5,0,0,3
1060 DATA 10,7,1,-1,-1,-4,0,-1,
1,0,4,1,1,4,0,1,-1,0,-1,-1,0
1070 DATA 6,0,0,0,6,0,-3,6,0,0,
3,0,-6
1080 DATA 6,0,0,6,0,-3,0,0,6,-3
,0,6,0
1090 DATA 7,0,0,6,0,-3,0,0,5,-1
,1,-1,0,-1,-1
1100 DATA 6,0,0,0,6,0,-3,6,3,-6
,-3,6,-3
1110 DATA 3,0,0,0,6,6,0
1120 DATA 5,0,6,0,-6,3,3,3,-3,0
,6
1130 DATA 4,0,6,0,-6,6,6,0,-6
1140 DATA 9,1,0,-1,1,0,4,1,1,4,
0,1,-1,0,-4,-1,-1,-4,0
1150 DATA 7,0,6,0,-6,5,0,1,1,0,
1,-1,1,-5,0
1160 DATA 11,5,0,-4,0,-1,1,0,4,
1,1,4,0,1,-1,-2,-1,2,1,0,-4,-1,
-1
1170 DATA 9,0,6,0,-6,5,0,1,1,0,
1,-1,1,-5,0,4,0,2,3
1180 DATA 12,6,1,-1,-1,-4,0,-1,
1,0,1,1,1,4,0,1,1,0,1,-1,1,-4,0
,-1,-1
1190 DATA 4,3,6,0,-6,-3,0,6,0
1200 DATA 6,0,0,0,5,1,1,4,0,1,-
1,0,-5
1210 DATA 3,0,0,3,6,3,-6
1220 DATA 5,0,0,1,6,2,-3,2,3,1,
-6
1230 DATA 5,0,0,6,6,-3,-3,-3,3,
6,-6
1240 DATA 5,3,6,0,-3,-3,-3,3,3,
3,-3
1250 DATA 4,0,0,6,0,-6,6,6,0
```

```
10 SCREEN 0
20 DIM N(26),A(26,12,2)
30 FOR N=1 TO 26
40 READ N(N)
50 FOR M=1 TO N(N)
60 READ A(N,M,1),A(N,M,2)
70 NEXT M
75 NEXT N
80 INPUT"introduza a palavra";A
$
90 INPUT"introduza a escala do
x";X
100 INPUT"introduza a escala do
 y";Y
110 SCREEN 2
115 FOR N=1 TO LEN(A$)
120 T$=MID$(A$,N,1) IF T$<"A" O
R T$>"Z"THEN GOTO 80
130 J=(10*(N-1)*X)+X*A((ASC(T$)
-64),1,1):K=10+Y*A((ASC(T$)-64)
,1,2)
140 FOR M=2 TO N(ASC(T$)-64)
150 F=X*A((ASC(T$)-64),M,1)
160 G=Y*A((ASC(T$)-64),M,2)
170 LINE (J,K)-(J+F,K+G)
180 J=J+F:K=K+G
190 NEXT M
195 NEXT N
200 IF INKEY$="" THEN 200
210 GOTO 80
1000 DATA 9,0,6,0,-5,1,-1,4,0,1
,1,0,3,0,2,0,-3,-6,0
1010 DATA 12,0,0,0,6,5,0,1,-1,0
,-1,-1,-1,-5,0,5,0,1,-1,0,-1,-1
,-1,-5,0
1020 DATA 8,6,1,-1,-1,-4,0,-1,1
,0,4,1,1,4,0,1,-1
1030 DATA 7,0,0,0,6,4,0,2,-2,0,
-2,-2,-2,-4,0
1040 DATA 7,6,0,-6,0,0,6,6,0,-6
,0,0, 3,5,0
1050 DATA 7,0,0,6,0,-6,0,0,3,5,
0,-5,0,0,3
1060 DATA 10,7,1,-1,-1,-4,0,-1,
1,0,4,1,1,4,0,1,-1,0,-1,-1,0
1070 DATA 6,0,0,0,6,0,-3,6,0,0,
3,0,-6
1080 DATA 6,0,0,6,0,-3,0,0,6,-3
,0,6,0
1090 DATA 7,0,0,6,0,-3,0,0,5,-1
,1,-1,0,-1,-1
1100 DATA 6,0,0,0,6,0,-3,6,3,-6
,-3,6,-3
1110 DATA 3,0,0,0,6,6,0
1120 DATA 5,0,6,0,-6,3,3,3,-3,0
,6
1130 DATA 4,0,6,0,-6,6,6,0,-6
1140 DATA 9,1,0,-1,1,0,4,1,1,4,
0,1,-1,0,-4,-1,-1, 4,0
1150 DATA 7,0,6,0,-6,5,0,1,1,0,
1,-1,1,-5,0
1160 DATA 11,5,0,-4,0,-1,1,0,4,
1,1,4,0,1,-1,-2,-1,2,1,0,-4,-1,
-1
1170 DATA 9,0,6,0,-6,5,0,1,1,0,
1,-1,1,-5,0,4,0,2,3
1180 DATA 12,6,1,-1,-1,-4,0,-1,
1,0,1,1,1,4,0,1,1,0,1,-1,1,-4,0
,-1,-1
1190 DATA 4,3,6,0,-6,-3,0,6,0
1200 DATA 6,0,0,0,5,1,1,4,0,1,-
1,0,-5
1210 DATA 3,0,0,3,6,3,-6
1220 DATA 5,0,0,1,6,2,-3,2,3,1,
-6
1230 DATA 5,0,0,6,6,-3,-3,-3,3,
6,-6
1240 DATA 5,3,6,0,-3,-3,-3,3,3,
3,-3
1250 DATA 4,0,0,6,0,-6,6,6,0
```

```
5 LOMEM: 16384
10  HOME : HGR : HCOLOR= 3
20  DIM N(26),A(26,12,2)
30  FOR N = 1 TO 26
40  READ N(N)
50  FOR M = 1 TO N(N)
60  READ A(N,M,1),A(N,M,2)
```

```
70  NEXT M: NEXT N
75  HOME : HGR
80  VTAB 22: INPUT "INTRODUZA A
 PALAVRA ";A$: IF A$ = "" THEN
 GOTO 80
90  INPUT "QUAL O FATOR X? ";X
100  INPUT "QUAL O FATOR Y? ";Y

110  FOR N = 1 TO  LEN (A$)
120 T$ =  MID$ (A$,N,1): IF T$
< "A" OR T$ > "Z" THEN  NEXT N:
 GOTO 80
130 J = (10 * (N - 1) * X) + X
* A(( ASC (T$) - 64),1,1):K = 1
0 + Y * A( ASC (T$) - 64,1,2)
140  FOR M = 2 TO N( ASC (T$) -
 64)
150 F = X * A(( ASC (T$) - 64),
M,1)
160 G = Y * A(( ASC (T$) - 64),
M,2)
170  HPLOT J,K TO J + F,G + K
180 J = J + F:K = K + G
190  NEXT M: NEXT N
200  GET R$
210  GOTO 75
1000  DATA  9,0,6,0,-5,1,-1,4,0
,1,1,0,3,0,2,0,-3,-6,0
1010  DATA  12,0,0,0,6,5,0,1,-1
,0,-1,-1,-1,-5,0,5,0,1,-1,0,-1,
-1,-1,-5,0
1020  DATA  8,6,1,-1,-1,-4,0,-1
,1,0,4,1,1,4,0,1,-1
1030  DATA  7,0,0,0,6,4,0,2,-2,
0,-2,-2,-2,-4,0
1040  DATA  7,6,0,-6,0,0,6,6,0,
-6,0,0,-3,5,0
1050  DATA  7,0,0,6,0,-6,0,0,3,
5,0,-5,0,0,3
1060  DATA  10,7,1,-1,-1,-4,0,-
1,1,0,4,1,1,4,0,1,-1,0,-1,-1,0
1070  DATA  6,0,0,0,6,0,-3,6,0,
0,3,0,-6
1080  DATA  6,0,0,6,0,-3,0,0,6,
-3,0,6,0
1090  DATA  7,0,0,6,0,-3,0,0,5,
-1,1,-1,0,-1,-1
1100  DATA  6,0,0,0,6,0,-3,6,3,
-6,-3,6,-3
1110  DATA  3,0,0,0,6,6,0
1120  DATA  5,0,6,0,-6,3,3,3,3
,0,6
1130  DATA  4,0,6,0,-6,6,6,0,-6

1140  DATA  9,1,0,-1,1,0,4,1,1,
4,0,1,-1,0,-4,-1,-1,-4,0
1150  DATA  7,0,6,0,-6,5,0,1,1,
0,1,-1,1,-5,0
1160  DATA  11,5,0,-4,0,-1,1,0,
4,1,1,4,0,1,-1,-2,-1,2,1,0,-4,-
1,-1
1170  DATA  9,0,6,0,-6,5,0,1,1,
0,1,-1,1,-5,0,4,0,2,3
1180  DATA  12,6,1,-1,-1,-4,0,-
1,1,0,1,1,1,4,0,1,1,0,1,-1,1,-4
,0,-1,-1
1190  DATA  4,3,6,0,-6,-3,0,6,0

1200  DATA  6,0,0,0,5,1,1,4,0,1
,-1,0,-5
1210  DATA  3,0,0,3,6,3,-6
1220  DATA  5,0,0,1,6,2,-3,2,3,
1,-6
1230  DATA  5,0,0,6,6,-3,-3,-3,
3,6,-6
1240  DATA  5,3,6,0,-3,-3,-3,3,
3,3,-3
1250  DATA  4,0,0,6,0,-6,6,6,0
```

Os programas aqui apresentados para as diferentes linhas de microcomputadores são muito semelhantes entre si. Em todos eles vamos encontrar 26 declarações **DATA**, uma para cada letra do alfabeto. Essas declarações contêm uma série de números que dizem ao computador como desenhar a letra, utilizando pequenas linhas.

O primeiro número da lista dá o total de linhas que entram na composição de cada letra — um L, por exemplo, requer menos linhas que um S. O número máximo usado é doze.

Os números seguintes são arranjados em pares, dando as coordenadas x e y de cada pequena seção da linha. Como já foi dito, tais números são relativos; assim, as linhas que serão efetivamente desenhadas dependem de fatores de escala. O primeiro par de números especifica o ponto de início da letra dentro de nossa caixa imaginária.

Os números contidos nas instruções **DATA** do programa são lidos para duas matrizes, **N** e **A**. **A** é uma matriz tridimensional; de 26 (número de letras) por 12 (número máximo de linhas) por 2 (vetores x e y).

COMO ESCOLHER UM CORDÃO

As linhas que permitem a entrada de palavras são semelhantes às do programa do artigo *Manchetes e Letreiros*. Elas verificam a entrada para evitar um cordão nulo; não existe, neste caso, nenhum limite para o número de caracteres utilizados.

Em seguida, o programa pede dois valores — um fator **X** e um fator **Y**. Esses valores determinam as reais dimensões de cada letra. Grosso modo, a escala 1 representa o tamanho padrão do caractere; 2 fornece altura ou largura dupla, enquanto 0.5, por sua vez, oferece metade do tamanho. Para valores menores que a unidade, podem-se obter efeitos estranhos. Uma vez que o computador não pode traçar uma fração de pixel, ele é obrigado a desenhar o pixel todo. Letras com um maior número de linhas apresentam mais facilidade para que isso ocorra. Desse modo, um S, por exemplo, pode terminar maior quando comparado com um T.

Se atribuirmos diferentes valores para cada fator, poderemos produzir interessantes variações nos caracteres, criando assim letras altas e magras, baixas e gordas etc.

Como não existe limite para o tamanho do cordão, é necessário tomar todo o cuidado no sentido de que o número de letras não ultrapasse as dimensões da tela. Quando isso ocorre, o computador emite uma mensagem de erro.

O cálculo de quantos caracteres cabem na tela para cada fator de escala é relativamente simples. Note que o valor importante é o fator X, que determina a largura da letra. Se você tiver um fator X igual a 2, só poderá colocar a metade do que seria possível com o fator 1. Com um fator 4, caberá apenas 1/4 dos caracteres.

ANALISE O CORDÃO

Assim como os outros programas que criavam letras, este usa um laço para verificar cada caractere do cordão. No Spectrum ele começa na linha 130; no TRS-Color, na linha 110.

O programa continua de maneira similar ao gerador de caracteres anterior, montando um cordão **T$** exatamente igual ao cordão original. Ao mesmo tempo, ele verifica se o caractere é uma letra de **A** a **Z**; qualquer outro caractere é tratado como se fosse um espaço. Se não for, o computador executará um **NEXT** para fazer avançar o laço **FOR...NEXT**. Se não houver mais letras, o programa voltará para que você possa dar entrada a mais palavras.

A ROTINA DE DESENHO

Quando o computador encontra uma letra no cordão, vai para a linha 150 (no Spectrum), que dá início à rotina de desenho. Nos outros computadores ela esta na linha 170. O conteúdo dessa linha, porém, difere de computador para computador, pois cada um deles tem um tamanho diferente de tela e conta com comandos distintos para desenhar em alta resolução.

A base para a rotina, entretanto, é a mesma para todos os micros. A variável de controle para o laço **FOR...NEXT**, **N**, é usada como um guia para a coordenada x da posição inicial; a ela são adicionados dois valores pelo computador. O primeiro é um valor relativo — a coordenada para desenho da matriz **A** —, enquanto o segundo é um valor que leva em conta o tamanho da tela e os fatores de escala.

Você já viu que um dos três elementos dessa matriz é usado para separar os detalhes das coordenadas x e y. Os outros dois elementos formam cada uma das 26 letras do alfabeto com até 24 números (lembre-se de que doze é o número máximo de linhas usadas por letra). A matriz **N** diz ao computador quantas linhas devem ser empregadas na definição de cada letra. Assim, quando o computador vai desenhar o caractere A, por exemplo, ele verifica inicialmente quantas linhas devem ser traçadas na matriz **N**. Esse número controla o laço para desenhar a quantidade exata de linhas de cada letra.

Os valores dos números contidos em **A**, a matriz principal do programa (tridimensional), equivalem ao código de caractere de cada letra menos 64 — esta é a razão pela qual as linhas 130 a 180 contêm expressões como **ASC(T$)-64** ou **CODE T$-64** ou **A-64**. Isso significa que o computador pode tomar o código de uma letra e usá-lo para contar linhas de desenho. Os valores relativos para x e y guardados na matriz são então multiplicados pelo fator de escala que você escolheu e passados para os comandos de desenho.

Assim que acaba de desenhar uma letra, o computador passa para a seguinte (caso haja) no cordão e o processo continua. Se as letras acabarem, o computador volta para a linha 80 (no Spectrum) ou 100 (nos outros micros) para que se possa entrar um novo cordão. Os microcomputadores das linhas TR-Color e MSX esperam que você tecle alguma coisa antes, visto que a tela de alta resolução tem que ser apagada para dar lugar à tela de texto.



Eis aqui as três versões para novas letras no Spectrum.

DESENHE SUAS PRÓPRIAS LETRAS

Você já viu três exemplos de tipos de letra. Inspirando-se neles, crie à vontade seus próprios caracteres. Os caracteres da ROM só podem ser expandidos, mas nada impede que as rotinas expansoras sejam utilizadas para trabalhar com seus caracteres redefinidos — uma explicação de como isso pode ser feito foi dada anteriormente no artigo *Conjunto de Blocos Gráficos* (1), à página 526.

As variações com os outros dois métodos são praticamente infinitas: você pode desenhar seus próprios caracteres a partir de blocos gráficos e guardá-los em uma matriz, como no programa do artigo *Manchetes e Letreiros*. Se não estiver satisfeito com o resultado, tente redesenhar tudo. Outra opção consiste em alterar a aparência das letras deste artigo, modificando os valores das linhas **DATA**.

USE OS NOVOS CARACTERES

Não é difícil criar letras maiores para serem mostradas na tela; mas, a não ser que se possa usá-las em algum programa, não há grande vantagem em se fazer isso. Se você optar pelas letras criadas neste programa ou no anterior, procure empregá-las em seus próprios programas.

Isso pode ser feito de várias maneiras. A mais comum incorpora o programa gerador de letras como uma sub-rotina do seu programa e chama-o sempre que for necessário.

Se você quiser usar o expansor de caracteres, essa será provavelmente a melhor solução. Mas, para o programa



Alterando-se os fatores X e Y, as letras podem ser traçadas em qualquer tamanho.



As telas mostram os tamanhos normal, extra-largo e extra-alto.

atual, ela está longe de ser razoável, uma vez que o programa toma muito espaço da memória e pouco sobra para seu próprio programa.

Como você verá em seguida, esse problema pode ser resolvido de várias maneiras: todas elas utilizam o programa para desenhar as letras, que são armazenadas de modo a estar disponíveis para uso posterior.

ARMAZENAGEM DOS DESENHOS

Uma dessas maneiras consiste em armazenar seus desenhos em gráficos UDG. Isso é fácil de fazer no TRS-Color, já que este possui o comando **GET**, que guarda o que está na tela. Embora com maiores dificuldades, o mesmo efeito pode ser obtido, em outros computadores, por meio do comando **POKE**.

É possível ainda gravar a parte da memória que contém os gráficos, como um bloco de código de máquina. Mais tarde, pode-se carregar toda a tela de volta para a memória. Entretanto, enquanto ela estiver na memória do micro, é preciso protegê-la, colocando-a acima ou abaixo do BASIC (para maiores detalhes, veja o artigo da página 526). Uma vez na memória, ela está em condições de ser utilizada.

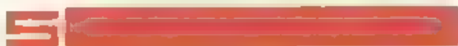
A melhor maneira de fazer isso é escrever uma rotina para ler o código da área protegida da RAM e passá-lo para a área de tela; ou, ainda, alterar o apontador do conjunto de caracteres, de modo que seu código se transforme em vários caracteres que poderão ser impressos. A rotina para mover o bloco de código da RAM para a área de tela seria algo como:

```
1000 FOR X=1 TO (COMPRIMENTO DO
 BLOCO DE CODIGO)
1010 POKE (ENDERECO DE INICIO D
A MEMORIA DE TELA)+X,PEEK (ENDE
RECO DE INICIO DO BLOCO DE CODI
GO)+X
1020 NEXT X
```

Esta, porém, não é a melhor solução para o problema, uma vez que levaria longo tempo para ser concluída. A dificuldade, felizmente, pode ser contornada por meio de uma rotina em linguagem de máquina que se encarregue de executar esse trabalho.

COMO GRAVAR A TELA

Uma solução alternativa consiste em gravar a tela e carregá-la juntamente com o programa. Nesse caso, o desenho permaneceria na tela até que você se decidisse a apagá-lo. Na maioria dos microcomputadores, esta constitui, sem dúvida, a melhor opção.



O Spectrum tem um comando especial que permite gravar em fita uma tela gráfica com grande facilidade. O desenho é gravado como um bloco de memória — processo descrito nas páginas 574 e 575; neste caso, porém, não é necessário especificar o endereço inicial e o tamanho do bloco. Basta usar **SCREEN$** logo após o nome de arquivo. Digitado no modo direto, esse comando grava uma figura com o nome "pic".

```
SAVE "pic" SCREEN$
```

Para carregar a figura:

```
LOAD "" SCREEN$
```

Pode-se, por exemplo, usar a tela como título para um jogo. Carrega-se primeiro a tela e a seguir o programa. Assim, ela ficará visível enquanto o programa estiver sendo carregado:

```
10 LOAD "" SCREEN$
20 LOAD "JOGO"
```

Grave agora com o comando:

```
SAVE "CARREG" LINE10
```

de modo que ele será executado automaticamente. O jogo será gravado com

```
SAVE "JOGO" LINE10
```

de modo que ele será executado automaticamente também. É importante lembrar que os programas têm que ser gravados na ordem correta na fita.



Para gravar uma tela em fita, utilize o comando:

```
CSAVEM"nomearquivo",1536,7679,
35725
```

Os primeiros dois números são os endereços inicial e final da área de gráficos (página 1 de qualquer **PMODE**, a não ser que se tenha um drive). **CLOADM** serve para carregar a figura de volta para a memória.

Essa tela gravada pode ser usada como uma página-título para um jogo; para isso, é preciso adicionar uma rotina de carregamento no início:

```
1 PMODE1,1 PCLS:SCREEN1,0
2 CLOADM
3 FOR D=1 TO 1000.NEXT
```

e certificar-se de que a figura será gravada logo após o jogo. Isso funcionará enquanto o programa não reduzir o número de páginas gráficas disponíveis inicialmente (neste caso, a figura poderia sobrepor-se ao programa).



O BASIC é muito lento e eu não gosto de programar em código de máquina. Tenho outras opções?

Existe uma família de linguagens que, digamos, estão "a meio caminho". Elas são compiladas, e não interpretadas. Como vimos anteriormente, o que toma tempo no BASIC é a interpretação de cada comando toda vez que o programa é rodado.

As linguagens compiladas usam um outro tipo de programa — parecido com os Assembler empregados na programação em código de máquina que converte a linguagem de alto nível para código de máquina. Assim, depois da compilação, estaremos rodando um programa em código de máquina, e não em linguagem de alto nível.

CANETAS ÓPTICAS

Espécie muito particular de ponteiro eletrônico, as canetas ópticas servem para fazer gráficos na tela e para selecionar itens de um menu. Veja como trabalhar com elas.



Comunicar-se com um computador geralmente significa recorrer ao teclado para digitar informações. Esta, porém, não é a única forma de comunicação com a máquina. Além dela, existe uma série de métodos alternativos. Provavelmente, o primeiro lugar dessa série pertence ao joystick, que pode ser usado tanto em jogos como em aplicações mais "sérias". Outro método igualmente barato e que apresenta nítida vantagem é o da caneta óptica.

As canetas ópticas, como o próprio nome sugere, têm grande semelhança com uma caneta comum, pois funcionam com base em movimentos de "desenhar" ou de apontar para coisas sobre a tela. Como são capazes de detectar o ponto da tela para o qual estão sendo apontadas, elas têm uma enorme série de aplicações potenciais, como, por exemplo, selecionar itens de um menu mostrado na tela e acionar as funções de um programa. Caso fosse utilizar o teclado em situações como essas, o usuário seria obrigado a afastar os olhos da tela e pressionar uma ou mais teclas. Com uma caneta óptica, porém, tudo fica bem mais rápido e simples: basta apontar para a opção.

APLICAÇÕES

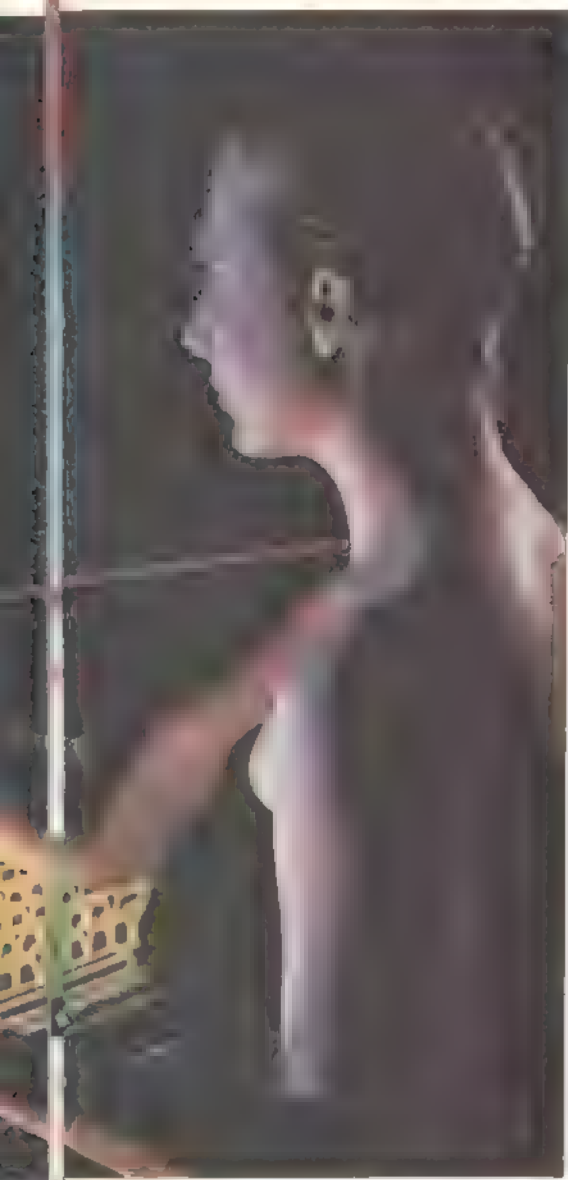
As canetas ópticas são particularmente apropriadas para aquelas aplicações em que o usuário precisa localizar rapidamente pontos espalhados pela tela, como quando deseja entrar as coordenadas de um ou mais pontos, colocar uma marca, desenhar uma reta, selecionar um menu etc.

Precisas, rápidas e de fácil manipulação, as canetas ópticas se prestam muito bem a aplicações gráficas. Nesse tipo de aplicação, o emprego das teclas para controle do cursor gráfico (ou, pior ainda, a utilização das teclas numéricas para entrar coordenadas) implicaria num lento e laborioso processo. Já com uma caneta óptica pode-se desenhar literalmente à mão livre: você traça apenas os contornos com a caneta, e o resultado aparece diretamente na tela. Outra técnica consiste em compor um desenho, usando a caneta para apontar para um menu com símbolos gráficos, cores, sombreados etc.

A caneta óptica mostra-se também de grande valia em programas de aplicações mais sérias, como processamento de textos, ou contabilidade (do tipo "você obtém o que vê"). Isso significa que, em vez de entrar instruções pelo teclado, elas são entradas por intermédio de uma caneta óptica que aponta para as opções disponíveis.

Até recentemente, programas aplicativos que empregassem canetas ópticas eram desenvolvidos pelos próprios usuários. Atualmente, com o crescente interesse por esses periféricos, algumas empresas produtoras de software já estão oferecendo programas que usam especificamente a caneta óptica.

Alguns programas "artísticos" ou de projeto por computador possibilitam a criação de desenhos à base de figuras



geométricas (retas, círculos, elipses etc.), que podem ser preenchidas com cores. Muitos desses programas permitem também o uso de canetas ópticas para desenhos à mão livre sobre a tela. Com eles é possível elaborar trabalhos complexos e criativos.

Esse processo funciona da seguinte forma: um dos programas utiliza a caneta óptica para definir símbolos ou caracteres gráficos. Um segundo programa aproveita esses símbolos em projetos mais complexos. Aqui, a caneta óptica oferece nítidas vantagens sobre outros métodos de entrada gráfica e de seleção de opções.

Já é possível encontrar no mercado jogos que recorrem a canetas ópticas para que o jogador transporte figuras de um ponto para outro da tela. O xadrez e outros jogos de tabuleiro são bons exemplos desse emprego.

Palavras cruzadas, jogos de cartas, labirintos e quebra-cabeças variados completam o quadro de atividades lúdicas em que as canetas ópticas têm funções importantes. Essas funções são extensivas a jogos de aventuras, em que o jogador pode apontar para os objetos que deseja apanhar ou levar.

As canetas ópticas podem ainda assumir formas especiais, como o rifle óptico que serve para centrar a mira em alvos na tela. Esse rifle já existe tanto em micros domésticos quanto em certos videogames. Está, portanto, especificamente voltado para jogos.

COMO FUNCIONA A CANETA ÓPTICA

Uma caneta óptica é usada basicamente para detectar pontos luminosos presentes na tela por meio de um fotodetector. Quando ela é apontada para alguma locação na tela, ocorre um trabalho conjunto entre os circuitos eletrônicos de detecção na caneta e o software carregado no computador; o objetivo desse trabalho é calcular as coordenadas do ponto para o qual a caneta está sendo apontada.

Uma das técnicas desse processo baseia-se no modo como a imagem é produzida em um cinescópio, tubo de TV ou monitor. O feixe de elétrons que bombardeia a tela é movimentado de forma a deslocar um pequeno ponto de luz sobre ela em um padrão que vai da esquerda para a direita e de cima para baixo. Esse padrão tem cerca de seiscentas linhas de varredura, repetidas a cada 1/60 de segundo. Como a varredura é muito rápida, o olho humano percebe apenas uma imagem estável.

A caneta óptica tem na ponta um minúsculo sensor fotoelétrico (normalmente, um diodo ou transistor sensível à luz) que detecta o pontinho de luz quando este passa em seu foco. Um sinal elétrico é então produzido, amplificado e transmitido para o computador. Este, por sua vez, é que controla o movimento de varredura e, por isso, consegue calcular as coordenadas do ponto, com base no tempo que o feixe de elétrons leva para partir do topo e chegar ao ponto de detecção.

As canetas ópticas variam segundo o tipo de sensor luminoso empregado, onde ele é colocado e onde se localizam os circuitos eletrônicos associados; ou, ainda, conforme permitam ou não diferentes modos operacionais. Até mesmo a maneira pela qual mostram que o sinal foi detectado serve para diferenciar os vários tipos de caneta óptica. Finalmente, como não poderia deixar de ser, elas são classificadas de acordo com o modelo de computador com o qual são compatíveis.

Todos esses fatores afetam o preço e o desempenho da caneta óptica, bem como suas características operacionais. Dentre estas, as principais são a *sensibilidade* e a *resolução.*

SENSIBILIDADE

A sensibilidade de uma caneta óptica revela a gama de intensidades luminosas que ela é capaz de detectar na tela. As melhores canetas captam todas ou quase todas as cores exibidas. As de qualidade inferior detectam apenas as cores mais brilhantes ou as intensidades mais fortes. Para muitas aplicações, isso será mais do que suficiente, embora imponha uma severa limitação na flexibilidade de uso, particularmente em aplicações gráficas, onde o fundo é normalmente escuro.

Um problema freqüente com as canetas ópticas é a chamada detecção espúria. Esta ocorre quando a sensibilidade é muito aumentada pela excessiva amplificação do sinal proveniente do sensor de luminosidade. Esse aumento amplia a probabilidade de a caneta óptica ser ativada por outras fontes de luz, tais como reflexos na tela ou a própria luz ambiente.

Para enfrentar esse problema, as canetas ópticas são equipadas com mecanismos destinados a impedir a ocorrência de detecções espúrias. Esses mecanismos podem ser constituídos, por exemplo, de células fotoelétricas e filtros de luz em proporções tais que se combinem com as características de cor do ponto na tela. Além disso, podem ser usados filtros eletrônicos para assegurar que a luz provenha realmente da tela (e não de uma fonte externa). Isso é possível porque os pontos na tela oscilam com uma freqüência característica (sessenta vezes por segundo). Assim, um circuito eletrônico simples (chamado "filtro passa-alto") é usado para selecionar apenas a luz que pisca com essa freqüência.

CANETAS DE ALTA RESOLUÇÃO

Uma vez que os problemas da sensibilidade e da detecção espúria tenham sido tecnicamente resolvidos, uma importante consideração adicional a fazer pode ser a resolução. Esta diz respeito à área mínima na tela que a caneta é capaz de detectar, que varia desde um único pixel na tela, nas melhores canetas ópticas, até um grupo inteiro de caracteres, nas piores.

A resolução de uma caneta depende do sensor que ela usa, da velocidade de reação e do método de colimação, ou de coleção da luz. Algumas canetas ópticas utilizam um tubo negro opaco para canalizar a luz até o sensor; outras empregam um sistema de lentes, ou pedaços curtos de fibra óptica. A alta resolução é importante se se quer desenhar com a pena, e não somente detectar posições. Uma caneta óptica com uma boa colimação conseguirá discriminar um ponto menor, e, conseqüentemente, será mais acurada.

Os modelos de canetas ópticas diferem quanto à quantidade e qualidade da documentação e do software que as acompanham. Para funcionarem a contento, elas devem contar com um bom suporte por parte de rotinas de software. A programação é razoavelmente simples, qualquer que seja o tipo de caneta ou de computador ao qual ela está ligada. Alguns fabricantes adicionam uma fita cassete ou disco com alguns programas de exemplo, enquanto outros apenas fornecem listagens de rotinas.

COMO ESCOLHER UMA CANETA ÓPTICA

Há vários fatores a considerar na hora de se escolher uma caneta óptica. Por exemplo, a precisão da caneta. Embora algumas canetas tenham uma alta resolução nominal, isso não quer dizer que elas sejam capazes de conservar a precisão por muito tempo. Algumas canetas ópticas são tão imprecisas que uma reta traçada com elas sobre o vídeo aparecerá como uma série aleatória de pontos acesos e apagados.

Geralmente, o preço desse periférico é proporcional à qualidade. Alguns dos mais caros, por exemplo, têm dois dispositivos de que não falamos ainda: o primeiro serve para anunciar que a caneta detectou um ponto de luz válido na tela; o segundo controla o sinal a ser enviado ao computador.

Um diodo emissor de luz (LED) é um



equipamento usado em algumas canetas para indicar se houve detecção correta: ele não é essencial para desenhar, mas pode ser útil quando se usa a caneta para escolher opções na tela ou detectar posições.

O controle do sinal é normalmente exercido por um botão ou um interruptor, situado quase sempre no corpo da caneta, ao qual o usuário recorre quando quer enviar a informação ao computador. Ele pode ser acionado com duas finalidades: para reduzir a chance de detecções falsas ou espúrias, e para ter mais controle sobre que leituras devem ser feitas pela caneta.

Existem vários tipos de interruptores: nos modelos mais sofisticados, eles podem tomar a forma de um pequeno contato na ponta da caneta, de tal maneira que esta será ativada assim que for encostada na tela.



COMPATIBILIDADE

Antes de comprar uma caneta óptica, convém testar se ela funcionará com o seu tipo de computador. Como acontece com muitos outros periféricos, as canetas ópticas costumam funcionar apenas quando ligadas ao computador para o qual foram feitas.

Algumas vezes, você poderá resolver esse problema adquirindo uma interface especial para o seu computador.

Não se esqueça de que esse periférico precisa de software para funcionar: se você comprar uma caneta óptica que não sirva especificamente para o seu micro, provavelmente passará por maus momentos, tentando descobrir como ela funciona.

PROBLEMAS COM CANETAS ÓPTICAS

A limpeza é um item essencial para a manutenção das canetas ópticas. Como elas trabalham com luz em intensidades muito baixas, qualquer sujeira na ponta da caneta ou na tela tende a interferir no sinal luminoso detectado. Para evitar esse tipo de problema, limpe regularmente a caneta de seu micro com um produto de limpeza semelhante ao usado para gravadores.

Outro problema peculiar das canetas ópticas é que a sua proximidade tende a obscurecer uma parte da tela. Esse inconveniente, porém, só se tornará realmente grave se a tela estiver coberta de informações.

Antes de comprar uma caneta óptica, assegure-se de que nenhum outro periférico — como joysticks e tabletes digitalizadores — é mais apropriado para o que pretende fazer. Não se esqueça de que usar uma caneta óptica por longo tempo pode ser muito desconfortável, devido à posição vertical forçada em que ela deve ser mantida.

O mercado de micros tem se caracterizado por um forte crescimento no emprego de aplicações em que o usuário deve interagir com a tela. Essa tendência provocará certamente a disseminação das canetas ópticas. Com isso, seu preço tenderá a baixar, e os programas capazes de utilizá-las se multiplicarão. Para o usuário que está procurando um acessório barato e diferente para o seu micro, a caneta óptica pode ser, assim, a aquisição ideal.

ACELERE SEUS PROGRAMAS

Você prefere programar em BASIC, mas lamenta a morosidade do computador em executar certas tarefas? Não desanime. Como verá neste artigo, existem várias maneiras de acelerar seus programas.

A princípio, todos os computadores parecem extremamente rápidos, executando num piscar de olhos tarefas que consumiriam muitas horas de um ser humano. Mas, ao escrever um jogo de ação, um programa que contenha muitos cálculos ou até mesmo uma ordenação complicada usando o BASIC, o usuário começa a se decepcionar. Realmente a máquina demora para executar certas tarefas, e, depois de alguma experiência, é provável que você também acabe reclamando de sua lentidão...

Os programas mais rápidos são os escritos em código de máquina ou em linguagem Assembly, mas muitas pessoas optam pela facilidade da programação em BASIC. Um programa em BASIC, porém, nunca terá a mesma velocidade de execução de um programa em código de máquina, pois o computador gasta muito tempo traduzindo-o. O computador tem, já embutido como parte de seu hardware, um programa especial — o *interpretador* —, responsável pela tradução do BASIC para código de máquina.

Para os que não querem ou não gostam de escrever programas em código de máquina, mas ainda assim dão importância à velocidade e desejam extrair o máximo de rapidez do BASIC, temos algumas dicas. É importante, em primeiro lugar, tentar estruturar os programas adequadamente. Depois, deve-se selecionar os elementos do BASIC que o interpretador leva menos tempo para traduzir, de modo que cada linha de programa opere com velocidade máxima.

Cada máquina tem suas próprias limitações e, em conseqüência, cada programa requer um tratamento específico. Assim, não existem regras práticas que garantam a elaboração de programas perfeitos, mas, observando-se algumas recomendações, será possível torná-los bem mais eficientes.

CRONOMETRAGEM DO BASIC

Quase todas as linhas de microcomputadores possuem um temporizador interno que pode ser utilizado para comparar a velocidade de execução de programas. A rotina abaixo, que usaremos com exemplos dados mais adiante, permitirá que você observe as diferenças de velocidade existentes entre as várias formas de programação em BASIC. Os usuários do Apple precisarão recorrer a um cronômetro auxiliar de mão para fazer as comparações, pois os computadores dessa linha não dispõem de temporizador interno.

```
  1 POKE 23672,0: POKE 23673,0
  : POKE 23674,0
100 REM timer
120 FOR i=1 TO 100
130 GOSUB 200  NEXT i
140 LET b=PEEK 23672+256*PEEK
23673+65536*PEEK 23674
150 PRINT AT 5,5;(b-41)/5;"MIL
ISEGUNDOS"
160 STOP
200 REM
500 RETURN
```

```
100 TIME=0
110 FORK=1TO600·GOSUB200·NEXT
120 T=TIME+10
130 CLS:PRINTUSING" TEMPO USADO
 =## ### SEGUNDOS";T/60
140 END
200 REM
1000 RETURN
```



- VELOCIDADE DE EXECUÇAO EM PROGRAMAS BASIC
- CRONOMETRAGEM DOS COMANDOS
- RELAÇÃO ENTRE ESTRUTURA,
- MEMÓRIA E VELOCIDADE
- ARMAZENAGEM DAS VARIÁVEIS
- FUNÇÕES MATEMÁTICAS
- MULTIPLICAÇÃO E DIVISÃO
- ORDENAÇAO E BUSCA

```
10  REM  <USE UM CRONOMETRO
20  REM   PARA MEDIR O TEMPO>
100  [illegible]
110  PRINT  CHR$ (7)
120  PRINT : PRINT "INICIO"
130  FOR K = 1 TO 500
140  GOSUB 200
150  NEXT
160  PRINT  CHR$ (7)
170  PRINT : PRINT "FIM"
180  END
200  [illegible]
1000  RETURN
```

```
100 TIMER=0
110 FOR K=1 TO 600:GOSUB 200:NE
XT
120 T=TIMER-170
130 CLS:PRINT" TEMPO USADO = ";
T/60;"SEGUNDOS"
140 END
200 REM
1000 RETURN
```

Em todos os programas, o comando **REM** da linha 200 será substituído, mais tarde, pelo nosso teste. No programa do Spectrum, esse comando aparece também na linha 100, devendo ali permanecer, já que faz parte da calibração do temporizador interno.

ESTRUTURA

Como a estruturação do programa desempenha um papel fundamental na velocidade de sua execução, convém recapitular aqui algumas regras básicas.

Todas as sub-rotinas usadas com mais freqüência devem ser posicionadas no início do programa, já que o interpretador procura cada linha requisitada por um **GOSUB** a partir do começo da listagem. Assim, quanto menor for o número da linha de uma sub-rotina, mais depressa ela será encontrada pelo interpretador. Você pode argumentar que um atraso de milissegundos não altera nada no programa. Mas a economia de alguns milissegundos aqui e ali acabará fazendo uma boa diferença.

Existem programas mal elaborados, que não passam de um emaranhado de comandos **GOTO** usados indiscriminadamente. Esse tipo de programa — às vezes chamado de "programa-espaguete" —, além de ser de difícil compreensão, perde muito na velocidade de execução. Ao contrário, quando a disposição das rotinas é bem planejada, a execução do programa torna-se bem mais rápida.

MEMÓRIA E VELOCIDADE

Comumente, os programas mais curtos são também os mais rápidos. Porém, os três objetivos do programador — velocidade, clareza e economia de memória — quase sempre estão em conflito. O uso de vários comandos numa só linha, por exemplo, économiza memória e acelera o programa, mas dificulta sua compreensão e eventuais correções.

Um programa que trabalha com velocidade máxima geralmente é mais longo e utiliza mais memória. A maioria das técnicas para se economizar memória traz prejuízo para a velocidade. É o caso do emprego de sub-rotinas.

DICAS PARA ACELERAR SEU BASIC

Nas matrizes, comece com o índice 0 e não com 1.



Defina as matrizes no início de cada programa.



Se possível, use variáveis inteiras em vez de variáveis de ponto flutuante.



Nos laços **FOR...NEXT**, não coloque uma variável depois do **NEXT**.



Use variáveis no lugar de números.



Dê nomes curtos às variáveis.



Coloque as sub-rotinas de uso mais freqüente no começo do programa.



Use laços **FOR...NEXT** no lugar de laços do tipo:

```
100 LET X=X+1 : IF  X<20  THEN
GOTO 100
```



Numere as linhas com números mais baixos. Comece na linha 10 em vez de começar na linha 1000.



Use rotinas em código de máquina sempre que for possível. Torne-se um colecionador de sub-rotinas publicadas.

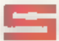


Nos programas voltados para cálculos, procure definir uma função que execute os cálculos repetitivos. Por exemplo.

```
10 DEF FN A(X)=X-(INT(X/360)*
360)
20 LET NUMERO=FN A(NUMERO)
```

é bem melhor que:

```
10 IF NUMERO>360 THEN LET NUMER
O=NUMERO 360: GOTO 10
```



Habitue-se a planejar seus programas. Além de erros, você evitará comandos **GOTO** e **GOSUB** desnecessários, que só comprometem a velocidade de execução.



Sub-rotinas economizam memória mas diminuem a rapidez do programa.



Remova todos os denominadores comuns. Por exemplo, troque:

```
10 LET X=Y/100+Z/100
```

por

```
10 LET X=(Y+Z)/100
```



Evite usar comandos **GOTO** sempre que possível.



Procure utilizar vários comandos na mesma linha. Perde-se em clareza, mas ganha-se em velocidade e economia.



Remova todos os espaços em branco e comandos **REM** desnecessários.



Ao usar um **IF...THEN**, coloque primeiro a condição que mais tende a ser FALSA.



Elas asseguram uma razoável economia de memória mas chamá-las toma tempo, o que não agrada aos programadores que buscam velocidade de execução. Como vimos anteriormente, o uso de sub-rotinas exige certo cuidado com a estruturação do programa. Sem isso, é impossível recuperar a velocidade que se perde ao chamá-las.

Uma linha como **LET A = VAL"100"** exemplifica bem a contradição que existe entre velocidade e economia de memória. Essa forma economiza memória, sem dúvida, mas, por outro lado, é de lenta execução se for comparada à forma mais comum **LET A = 100**

VARIÁVEIS

Saber como as variáveis são armazenadas na memória pode ser de grande utilidade para quem pretende acelerar a execução de programas. As variáveis são criadas à medida que aparecem no progama, sendo possível limpar a região em que se situam por meio dos comandos **RUN** ou **CLEAR**. Na maioria dos casos, uma nova variável provoca a ampliação dessa região para cima — o que não se aplica ao Sinclair Spectrum, que trabalha com as variáveis de uma maneira diferente.

Se criarmos, por exemplo, uma variável-cadeia e, depois, uma matriz numérica, todas as variáveis que se seguirem à matriz precisarão ser deslocadas para a parte superior da memória quando se acrescentar algo à cadeia.

Observe este programa:

```
100 LET T$=""
110 DIM A(1000)
120 LET T$=T$+"ACELERAR"
```

No BASIC da maioria dos micros, muito tempo de execução seria economizado se invertêssemos a ordem das duas primeiras linhas, ou seja, se a matriz fosse dimensionada antes de se definir a cadeia. Como está, o programa faz com que 5000 bytes sejam deslocados cada vez que se adiciona a palavra "ACELERAR" à variável-cadeia **T$**. Os usuários do Spectrum não precisam se preocupar, pois, como foi dito, esse micro trabalha com as variáveis de uma maneira diferente, não apresentando esse tipo de problema.

O uso de variável no lugar de números também assegura uma considerável economia de tempo. Essa regra aplica-se a todos os micros, menos ao Spectrum. No Apple, por exemplo, cada variável economiza cerca de cinco a dez milissegundos. Pode não parecer muito, mas imagine o que isso significa quan-

do se lida com um laço que é executado várias vezes. Já no Spectrum ocorre o oposto: a execução de um comando **LET C = 10 + 10** leva três milissegundos, enquanto a de **LET C = D + D** (onde D = 10) leva 4,2 milissegundos.

FUNÇÕES MATEMÁTICAS

Use a rotina apresentada anteriormente — que passaremos a chamar "cronômetro" — para testar estas duas formas de se obter a função potência.

```
200 LET C=4*4*4*4
```

ou, então:

```
200 LET C=4^4
```

É de se esperar que a função potência própria da máquina seja a mais rápida das duas. No Spectrum, por exemplo, a primeira forma leva seis milissegundos, e a segunda, 114 milissegundos. Estamos novamente diante de um caso em que a economia de memória (oferecida pela segunda forma) está em conflito com a velocidade.

Algumas vezes, lembramos que, no Spectrum, linhas do tipo

```
210 IF X>Y THEN LET Y=Y+1
220 IF X<Y THEN LET Y=Y-1
```

são executadas com maior velocidade quando substituídas por

```
10 LET Y = Y + (X>Y) - (X<Y)
```

O BASIC da linha Sinclair realmente permite esse tipo de comparação e programar assim é, sem dúvida, mais elegante. Mas, usando nossa rotina-cronômetro, podemos observar que a linha dupla é mais rápida que a simples.

Se você usa outro micro, teste o equivalente em sua máquina.



```
210 IF X>Y THEN Y=Y+1
220 IF X<Y THEN Y=Y-1
```

ou

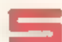
```
210 Y=Y+(X<Y)-(X>Y)
```

Forneça, em cada caso, valores cabíveis para **X** e **Y**.

MULTIPLICAÇÃO E DIVISÃO

As expressões **C=D*0.5** e **C=D/2** executam exatamente o mesmo cálculo, mas, como mostram os testes, a multiplicação é mais rápida que a divisão.

Faça um teste com as sugestões apresentadas a seguir e tome nota dos resultados para referências futuras.



```
200 LET C=10+10
200 LET C=D+D  (ONDE D=10)
200 LET C=10*10
200 LET C=10/10
200 LET C=10+PI
200 LET C=SIN 10
200 LET C=COS 10
200 LET C=TAN 10
200 LET C=VAL "10"
200 LET C=10
200 LET C=D  (ONDE D=10)
200 PRINT AT 21,0;"TESTE"
200 PRINT AT 21,0;A$ (A$=TESTE)
200 PRINT AT 21,0; 10+1000+500+
 5.5
200 PRINT AT 21,0;D+E+F+G (ON-
DE D=10,E+1000,ETC)
```



```
200 C=10+10
200 C=D+D  (onde D=10)
200 C=10*10
200 C=10/10
200 C=10+PI
200 C=SIN(10)
200 C=COS(10)
200 C=TAN(10)
200 C=VAL("10")
200 C=10
200 C=D  (onde D=10)
200 PRINT"TESTE"
200 PRINT A$  (onde A$="TESTE")
200 PRINT 10+1000+500+5.5
200 PRINT D+E+F+G  (onde D=10,E
=1000,etc)
```



```
200 C=10+10
200 C=D+D  (ONDE D=10)
200 C=10*10
200 C=10/10
200 C=10+PI
200 C=SIN(10)
200 C=COS(10)
200 C=TAN(10)
200 C=VAL("10")
200 C=10
200 C=D  (ONDE D=10)
200 PRINT"TESTE"
200 PRINTA$  (ONDE A$="TESTE")
200 PRINT 10+1000+500+5.5
200 PRINT D+E+F+G  (ONDE D=10,E
=1000,ETC)
```



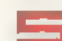
```
200 C=10+10
200 C=D+D  (ONDE D=10)
200 C=10*10
200 C=10/10
200 C=10+PI
200 C=SIN(10)
200 C=COS(10)
200 C=TAN(10)
200 C=VAL("10")
200 C=10
200 C=D  (ONDE D=10)
200 PRINT @260,"TESTE"
200 PRINT @260,A$ (A$="TESTE")
200 PRINT @260,10+1000+500+5.5
200 PRINT @260,D+E+F+G  (ONDE D
=10, E=1000, ETC)
```

ORDENAÇÃO E BUSCA

Como as técnicas de ordenação já foram tema de artigos anteriores, não entraremos aqui em maiores detalhes. Como vimos, a ordenação tipo *Shell-Metzner* é a mais indicada quando se trata de manipular um número de itens superior a cem. Esse tipo de ordenação tem a característica de ser tanto mais rápido quanto maior for o número de itens. Porém, como era de se esperar, também nesse caso a velocidade envolve certo sacrifício de memória.

Ao contrário do que muitos pensam, ordenação e busca não são a mesma coisa. Buscar é acessar um dado (ou um conjunto de dados) o mais rapidamente possível; ordenar é colocar vários dados em ordem. Suponhamos que você precise encontrar o telefone de certa pessoa em uma lista armazenada no computador. Ora, não haveria vantagem nenhuma se o tempo gasto pelo computador para encontrar tal número fosse maior que o tempo necessário para achá-lo numa lista telefônica. Por isso, quando se trata de ordenar ou buscar, sempre se procura o método mais rápido.

A busca serial (listada a seguir) simula o procedimento de uma pessoa que procura determinado dado analisando item por item de uma lista.



```
10 DATA "ANTILOPE","CACHORRO"
,"ELEFANTE","GOLFINHO","JUMEN
TO","LEOPARDO","MACACO","PERI
QUITO","RAPOSA","SAPO"
20 RESTORE 10: DIM B$(10,9)
30 CLS : FOR i=1 TO 10: READ
B$(i): PRINT AT i,5;B$(i):
NEXT i
40 POKE 23658,8: INPUT "Qual
o animal ?";A$
50 FOR X=1 TO 10
60 IF B$(X, TO LEN A$)=A$
THEN  PRINT  FLASH 0;AT X,13;
""; FLASH 1;" ACHEI": GOTO 40
70 NEXT X
```



```
10 REM <busca serial>
20 DIM B$(10)
30 CLS:FORI=1TO10:READB$(I):LOC
ATE0,I+3:PRINTB$(I):NEXT
40 LOCATE0,17:INPUT"Qual a anim
al";A$
50 FORX=1TO10
60 IF B$(X)=A$ THEN LOCATE13,X+
3:PRINT"<< ACHEI":X=10
70 NEXT:GOTO40
```

```
80 DATA ANTILOPE,CACHORRO,ELEFA
NTE,GOLFINHO,JUMENTO,LEOPARDO,M
ACACO,PERIQUITO,RAPOSA,SAPO
```



```
10   REM  <BUSCA SERIAL>
20   DIM B$(10)
30   HOME : FOR I = 1 TO 10
35   READ B$(I): PRINT  TAB( 11)
;B$(I)
40   NEXT : PRINT : PRINT
45   INPUT "QUAL O ANIMAL ?";A$
50   FOR X = 1 TO 10
60   IF B$(X) = A$ THEN  VTAB (X
): PRINT "ACHEI >>":X = 10
70   NEXT : VTAB (13): GOTO 45
80   DATA  ANTILOPE,CACHORRO,ELE
FANTE,GOLFINHO,JUMENTO,LEOPARDO
,MACACO,PERIQUITO,RAPOSA,SAPO
```



```
10 REM BUSCA SERIAL
20 DIM B$(10)
30 CLS:FOR I=1 TO 10:READ B$(I)
:PRINT @33+I*32,B$(I);:NEXT
40 PRINT @417,STRING$(30,32);ST
RING$(30,8);:INPUT"QUAL O ANIMA
L ";A$
50 FOR X=1 TO 10
60 IF B$(X)=A$ THEN PRINT @46+X
*32,CHR$(191);"ACHEI";:X=10
70 NEXT:GOTO 40
80 DATA ANTILOPE,CACHORRO,ELEFA
NTE,GOLFINHO,JUMENTO,LEOPARDO,M
ACACO,PERIQUITO,RAPOSA,SAPO
```

A rotina de busca binária (listada a seguir), um pouco mais difícil de se programar que a de busca serial, é bem mais rápida que esta. A diferença de velocidade entre ambas torna-se maior à medida que a quantidade de dados da lista aumenta.

A busca binária é mais rápida que a serial porque não utiliza o processo de inspeção item por item. Ela requer que todos os dados tenham sido previamente colocados em ordem alfabética ou numérica. Cumprido esse pré-requisito, o computador analisa o elemento central da lista, para definir se se orientará para sua metade superior ou para sua metade inferior.

A metade escolhida será novamente dividida ao meio e, a partir da análise desse novo elemento central, outra metade será escolhida e igualmente dividida ao meio.

O processo se repete até que o item desejado seja encontrado, ou, caso contrário, até que não haja mais nenhuma possibilidade de dividir a lista ao meio.

No início, o computador não procura por itens que sejam totalmente equivalentes, mas apenas compara a primeira letra, verificando se, na ordem alfabética, ela vem antes ou depois da do item desejado.



```
10 CLS : RESTORE
20 LET t=10: LET b=1
30 DIM n$(10,10)
40 FOR c=1 TO 10
50 READ n$(c)
60 NEXT c
70 INPUT "NOME DO ANIMAL ";a$
75 LET a$=a$+"           "( TO
10-LEN a$)
80 PAUSE 50
95 CLS
100 IF n$(t)=a$ THEN  PRINT n$
(t),t: GOTO 200
110 IF n$(b)=a$ THEN  PRINT n$
(b),b: GOTO 200
120 LET p=INT (0.5+(t+b)/2)
130 IF n$(p)=a$ THEN  PRINT n$
(p),p: GOTO 200
140 IF n$(p)>a$ THEN  LET t=p
150 IF n$(p)<a$ THEN  LET b=p
160 IF t-b=1 THEN  PRINT "    N
AO ACHEI": GOTO 200
170 GOTO 100
200 IF INKEY$="" THEN  GOTO
 200
210 RUN
580 DATA "Antilope","Cachorro"
,"Elefante","Golfinho","Jument
o","Leopardo","Macaco","Periqu
ito","Raposa","Sapo"
```



```
10 CLS
20 T=10:B=1
30 DIM N$(10)
40 FOR C=1 TO 10
50 READ N$(C)
60 NEXT
70 INPUT"nome do animal";A$
80 TIME=0
90 IF TIME<50 THEN 90 ELSE CLS
100 IF N$(T)=A$ THEN PRINTN$(T)
,T:GOTO200
110 IF N$(B)=A$ THEN PRINTN$(B)
,B:GOTO200
120 P=INT(.5+(T+B)/2)
130 IF N$(P)=A$ THEN PRINTN$(P)
,P:GOTO200
140 IF N$(P)>A$ THEN T=P
150 IF N$(P)<A$ THEN B=P
160 IF T-B=1 THEN PRINT"    não
 achei":GOTO200
170 GOTO 100
200 IF INKEY$="" THEN 200
210 RUN
580 DATA ANTILOPE,CACHORRO,ELEF
ANTE,GOLFINHO,JUMENTO,LEOPARDO,
MACACO,PERIQUITO,RAPOSA,SAPO
```



```
10   HOME
20 T = 10:B = 1
30   DIM N$(10)
40   FOR C = 1 TO 10
50   READ N$(C)
60   NEXT
70   INPUT "QUAL O ANIMAL ?";A$
```



```
80   FOR TT = 0 TO 200: NEXT
100   IF N$(T) = A$ THEN  PRINT
N$(T),T: GOTO 200
110   IF N$(B) = A$ THEN  PRINT
N$(B),B: GOTO 200
120 P =  INT (.5 + (T + B) / 2)

130   IF N$(P) = A$ THEN  PRINT
N$(P),P: GOTO 200
140   IF N$(P) > A$ THEN T = P
150   IF N$(P) < A$ THEN B = P
160   IF T - B = 1 THEN  PRINT "
    NAO ACHEI": GOTO 200
170   GOTO 100
200   GET Z$: IF Z$ = "" THEN 20
0
210   RUN
580   DATA   ANTILOPE,CACHORRO,E
```



```
LEFANTE,GOLFINHO,JUMENTO,LEOPAR
DO,MACACO,PERIQUITO,RAPOSA,SAPO
```



```
10 CLS
20 T=10:B=1
30 DIM N$(10)
40 FOR C=1 TO 10
50 READ N$(C)
60 NEXT
70 INPUT" NOME DO ANIMAL ";A$
80 TIMER=0
90 IF TIMER<50 THEN 90 ELSE CLS
100 IF N$(T)=A$ THEN PRINT N$(T
),T:GOTO 200
110 IF N$(B)=A$ THEN PRINT N$(B
),B:GOTO 200
120 P=INT(.5+(T+B)/2)
130 IF N$(P)=A$ THEN PRINT N$(P
),P:GOTO 200
140 IF N$(P)>A$ THEN T=P
150 IF N$(P)<A$ THEN B=P
160 IF T-B=1 THEN PRINT"     NAO
 ACHEI":GOTO 200
170 GOTO 100
200 IF INKEY$="" THEN 200
210 RUN
580 DATA ANTILOPE,CACHORRO,ELEF
ANTE,GOLFINHO,JUMENTO,LEOOPARDO
,MACACO,PERIQUITO,RAPOSA,SAPO
```

Como foi dito, não existem regras práticas para a obtenção de maior velocidade de execução em nossos programas. Os melhores resultados são conseguidos dedicando-se atenção especial a detalhes que, isolados, parecem ser insignificantes, mas que, juntos, fazem a diferença. Alguns desses detalhes são lembrados no quadro da página 932. Observando as recomendações ali contidas, você certamente obterá alguma economia de tempo. Mas não se esqueça de que, muitas vezes, a rapidez de um programa procede de uma descoberta individual, resultante, até mesmo, de um simples acaso.

UM ASSISTENTE PARA O DOS

Se você já dispõe de um acionador de disquetes mas tem preguiça de aprender os comandos do sistema operacional (DOS), utilize um programa assistente e durma tranqüilo.

Qualquer que seja a aplicação a que se destine o seu microcomputador, convém adquirir um acionador de disquetes. Desse modo, você poderá dispor de um sistema bem mais confiável e rápido do que as fitas cassete.

FUNÇÕES DO DOS

Do ponto de vista operacional, contudo, esse sistema apresenta uma pequena desvantagem. Com ele, o usuário é obrigado a aprender um número adicional de comandos, específicos para operações com disquetes. Tais comandos fazem parte do chamado Sistema Operacional de Discos (DOS, sigla decorrente de seu nome em inglês). Eles funcionam mais ou menos da mesma maneira nos diferentes modelos de computador, variando apenas a forma com que são escritos, e a sua sintaxe de uso.

As funções básicas de um DOS são:
- examinar o conteúdo de um disquete (catálogo);
- copiar um arquivo de um disquete para outro ou fazer uma cópia com nome diferente no mesmo disquete;
- mudar o nome de um arquivo;
- apagar um arquivo do disquete;
- listar um arquivo na tela do computador ou na impressora;
- examinar qual o espaço que está disponível no disquete;
- proteger ou desproteger um arquivo contra apagamento ou modificação;
- executar programas em BASIC ou em linguagem de máquina etc.

Nem sempre, contudo, o usuário está disposto a aprender em detalhe todas essas funções. De certo modo, estas caracterizam uma "linguagem" de comandos, que em alguns computadores é parte das instruções do BASIC, enquanto em outros é um sistema totalmente separado, do qual o interpretador BASIC é apenas um programa a mais.

A listagem apresentada aqui serve como "assistente" para a operação dos vários comandos do DOS; ela dispensa a necessidade de aprendê-los em detalhe. Seu funcionamento se apóia em um "cardápio" de opções, mostrado na tela assim que é executado. Escolhendo uma das opções apresentadas, o programa automaticamente seleciona e executa o comando correspondente do DOS. Em muitos casos, será necessário entrar informações adicionais, tal como o nome de um arquivo. Possíveis erros cometidos pelo usuário, como tentar apagar um arquivo protegido, serão detectados pelo programa, que exibirá na tela a mensagem correspondente.

Neste artigo abordaremos versões para os computadores das linhas TRS-80, TRS-Color, MSX, Apple II e TK-2000. Os micros da linha Sinclair também podem ser conectados a disquetes, ou "microdrives", fitas em alça sem fim, de comportamento semelhante aos disquetes; mas como estes periféricos não são encontrados no país, deixamos de apresentar uma versão para eles.

Digite os programas a seguir e armazene-os em um disquete. Para testar as opções do menu sem correr o risco de perder algo importante, copie no disquete alguns arquivos e programas de que não vai precisar.

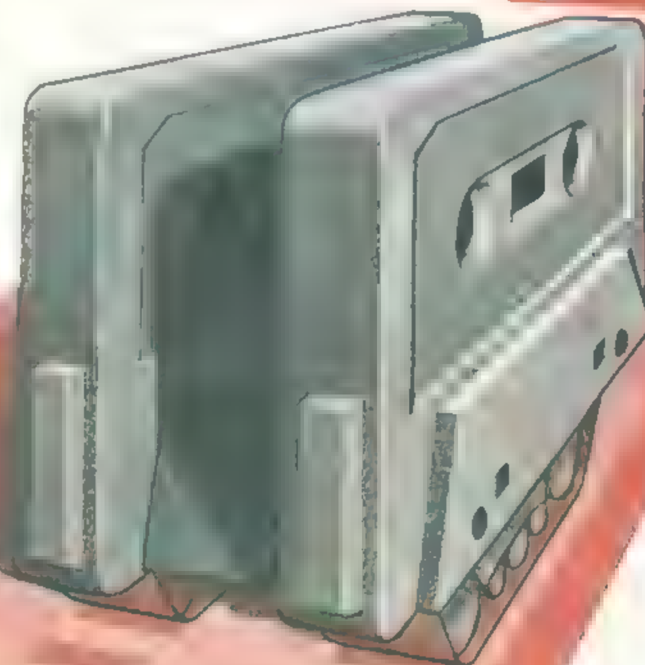
T

```
10 ON ERROR GOTO 960
20 GOSUB 950
30 PRINT:PRINT TAB(10) ; "ASSIS
TENTE D.O.S."
35 PRINT
```

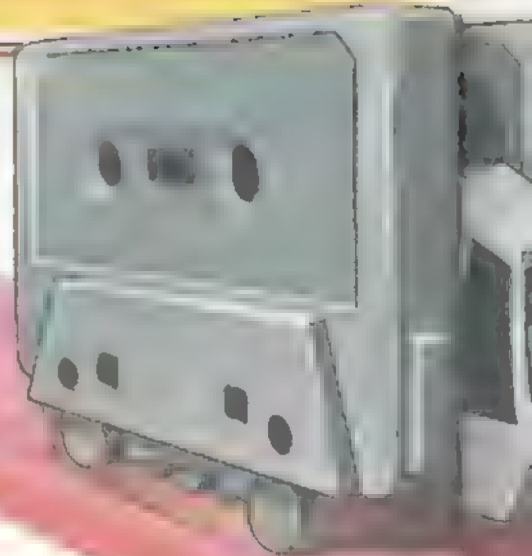


- O QUE É UM SISTEMA OPERACIONAL DE DISCOS
- FUNÇÕES PRINCIPAIS DO DOS
- UM ASSISTENTE INTELIGENTE
- INFORMAÇÕES ADICIONAIS
- DETECÇÃO DE ERROS
- COMO FUNCIONA
- O PROGRAMA
- EXIBIÇÃO DO MENU
- LIMITAÇÕES

```
40 PRINT "=====================
=========="
50 PRINT "<1>  Listar conteudo
do disco"
60 PRINT "<2>  Apagar um arquiv
o do disco"
70 PRINT "<3>  Mudar nome de um
arquivo"
80 PRINT "<4>  Executar um prog
rama"
90 PRINT "<5>  FIM DO PROGRAMA
190 PRINT "===================
========="
200 PRINT PRINT "Escolha uma op
cao ";
205 X$=INKEY$·IF X$="" THEN 205
210 OP=VAL(X$)
230 ON OP GOTO 250, 300, 350,
400, 900
240 GOTO 205
250 INPUT "DRIVE (0/1) ";D$
251 IF D$<>"0" AND D$<>"1" THEN
GOTO 250
252 GOSUB 950
255 PRINT TAB(10);"C A T A L O
G O"
260 PRINT:CMD "D:"+D$
270 GOSUB 910:RUN
300 PRINT.INPUT "Nome do arquiv
o para apagar";N$
310 KILL N$
320 PRINT:PRINT "Arquivo ";N$:"
apagado."
330 GOSUB 910·RUN
350 PRINT:INPUT "Nome do arquiv
o a ser mudado";N$
360 INPUT "Novo nome para o arq
```



```
uivo";N2$
370 NAME N$ AS N2$
380 PRINT:PRINT N$;" mudado par
a ";N2$
390 GOSUB 910:RUN
400 PRINT:INPUT "Nome do progra
ma ";N$
410 RUN N$
900 GOSUB 950
905 END
910 PRINT:PRINT:PRINT
915 PRINT "PRESSIONE QUALQUER
```



```
TECLA PARA CONTINUAR"
920 IF INKEY$="" THEN 920
925 RETURN
950 CLS:RETURN
960 E=ERR
965 IF E=51 THEN PRINT "*** ERR
O INTERNO":GOTO 995
970 IF E=53 THEN PRINT "*** ARQ
UIVO INEXISTENTE": GOTO 995
990 IF E=56 THEN PRINT "*** NOM
E ILEGAL DE ARQUIVO":GOTO 995
992 PRINT "*** ERRO DE SINTAXE"
995 GOSUB 910:RUN
```

```
10 ON ERROR GOTO 960
20 GOSUB 950
30 PRINT:PRINT TAB(10) ; "ASSIS
TENTE D.O.S."
35 PRINT
40 PRINT "--------------------
---------"
50 PRINT "<1>  LISTAR CONTEUDO
DO DISCO"
60 PRINT "<2>  APAGAR UM ARQUIV
O DO DISCO"
70 PRINT "<3>  MUDAR NOME DE UM
ARQUIVO"
80 PRINT "<4>  EXECUTAR UM PROG
[redacted]
90 PRINT "<5>  FIM DO PROGRAMA
190 PRINT "-------------------
---------"
200 PRINT:PRINT "ESCOLHA UMA OP
CAO:";
205 X$=INKEY$:IF X$="" THEN 205
210 OP=VAL(X$)
230 ON OP GOTO 250, 300, 350,
400, 900
240 GOTO 205
250 GOSUB 950
255 PRINT TAB(10);"C A T A L O
G O"
260 PRINT:DIR
270 GOSUB 910:RUN
300 PRINT:INPUT "NOME DO ARQUIV
O PARA APAGAR";N$
310 KILL N$
320 PRINT:PRINT "ARQUIVO ";N$;"
APAGADO."
330 GOSUB 910:RUN
350 PRINT:INPUT "NOME DO ARQUIV
O A SER MUDADO";N$
360 INPUT "NOVO NOME PARA O ARQ
UIVO";N2$
370 NAME N$ AS N2$
380 PRINT:PRINT N$;" MUDADO PAR
A ";N2$
390 GOSUB 910:RUN
400 PRINT:INPUT "NOME DO PROGRA
MA ";N$
410 RUN N$
900 GOSUB 950
905 END
910 PRINT:PRINT:PRINT
915 PRINT "PRESSIONE QUALQUER
TECLA PARA CONTINUAR"
920 IF INKEY$="" THEN 920
925 RETURN
950 CLS:RETURN
960 E=ERR
965 IF E=51 THEN PRINT "*** ERR
O INTERNO":GOTO 995
970 IF E=53 THEN PRINT "*** ARQ
UIVO INEXISTENTE": GOTO 995
990 IF E=56 THEN PRINT "*** NOM
E ILEGAL DE ARQUIVO":GOTO 995
992 PRINT "*** ERRO DE SINTAXE"
995 GOSUB 910:RUN
```

```
10 ON ERROR GOTO 960
20 GOSUB 950
30 PRINT:PRINT TAB(10) ; "ASSIS
TENTE D.O.S."
35 PRINT
40 PRINT "--------------------
---------"
50 PRINT "<1>  Listar conteudo
do disco"
60 PRINT "<2>  Apagar um arquiv
o do disco"
70 PRINT "<3>  Mudar nome de um
arquivo"
80 PRINT "<4>  Executar um prog
rama"
90 PRINT "<5>  FIM DO PROGRAMA
190 PRINT "-------------------
---------"
200 PRINT:PRINT "Escolha uma op
ção:";
205 X$=INKEY$:IF X$="" THEN 205
210 OP=VAL(X$)
230 ON OP GOTO 250, 300, 350,
400, 900
240 GOTO 205
250 GOSUB 950
255 PRINT TAB(10);"C A T A L O
G O"
260 PRINT:FILES
270 GOSUB 910:RUN
300 PRINT:INPUT "Nome do arquiv
o para apagar";N$
310 KILL N$
320 PRINT:PRINT "Arquivo ";N$;"
apagado."
330 GOSUB 910:RUN
350 PRINT:INPUT "Nome do arquiv
o a ser mudado";N$
360 INPUT "Novo nome para o arq
uivo";N2$
370 NAME N$ AS N2$
380 PRINT:PRINT N$;" mudado par
a ";N2$
390 GOSUB 910:RUN
400 PRINT:INPUT "Nome do progra
ma ";N$
410 RUN N$
900 GOSUB 950
905 END
910 PRINT:PRINT:PRINT
915 PRINT "PRESSIONE QUALQUER
TECLA PARA CONTINUAR"
920 IF INKEY$="" THEN 920
925 RETURN
950 CLS:RETURN
960 E=ERR
965 IF E=51 THEN PRINT "*** ERR
O INTERNO":GOTO 995
970 IF E=53 THEN PRINT "*** ARQ
UIVO INEXISTENTE": GOTO 995
990 IF E=56 THEN PRINT "*** NOM
E ILEGAL DE ARQUIVO":GOTO 995
992 PRINT "*** ERRO DE SINTAXE"
995 GOSUB 910:RUN
```

```
10 ON ERR GOTO 960
15 LET D$=CHR$(4)
20 GOSUB 950
30 PRINT:PRINT TAB(10) ; "ASSIS
TENTE D.O.S."
35 PRINT
40 PRINT "--------------------
---------"
50 PRINT "<1>  LISTAR CONTEUDO
DO DISCO"
```



```
60 PRINT "<2>  APAGAR UM ARQUIV
O DO DISCO"
65 PRINT "<3>  MUDAR NOME DE UM
ARQUIVO"
70 PRINT "<4>  EXECUTAR UM PROG
RAMA"
75 PRINT "<5>  PROTEGER UM ARQU
IVO"
80 PRINT "<6>  DESPROTEGER UM A
RQUIVO"
85 PRINT "<7>  VERIFICAR UM ARQ
UIVO"
90 PRINT "<8>  FIM DO PROGRAMA
190 PRINT "-------------------
---------"
200 PRINT:PRINT "ESCOLHA UMA OP
CAO:";
205 GET X$
210 LET OP=VAL(X$)
230 ON OP GOTO 250, 300, 350,
400, 450, 500, 550, 900
240 GOTO 205
250 GOSUB 950
255 PRINT TAB(10);"C A T A L O
G O"
260 PRINT:PRINT D$;"CATALOG"
270 GOSUB 910:RUN
300 PRINT:INPUT "NOME DO ARQUIV
O PARA APAGAR";N$
310 PRINT D$;"DELETE ";N$
320 PRINT:PRINT "ARQUIVO ";N$;"
APAGADO."
330 GOSUB 910:RUN
350 PRINT:INPUT "NOME DO ARQUIV
O A SER MUDADE";N$
360 INPUT "NOVO NOME PARA O ARQ
UIVO";N2$
370 PRINT D$;"RENAME
";N$;",";N2$
380 PRINT:PRINT N$;" MUDADO PAR
A ";N2$
390 GOSUB 910:RUN
400 PRINT:INPUT "NOME DO PROGRA
MA ";N$
410 PRINT D$;"RUN ";N$
450 PRINT:INPUT "NOME DO ARQUIV
O PARA PROTEGER ";N$
460 PRINT D$;"LOCK ";N$
470 PRINT:PRINT "ARQUIVO ";N$;"
PROTEGIDO."
480 GOSUB 910:RUN
500 PRINT:INPUT "NOME DO ARQUIV
O PARA DESPROTEGER ";N$
510 PRINT D$;"UNLOCK ";N$
520 PRINT:PRINT "ARQUIVO ";N$;"
DESPROTEGIDO."
530 GOSUB 910:RUN
550 PRINT:INPUT "NOME DO ARQUIV
O PARA VERIFICAR ";N$
560 PRINT D$;"VERIFY ";N$
570 PRINT:PRINT "ARQUIVO ";N$;"
CORRETO."
580 GOSUB 910:RUN
900 GOSUB 950
905 END
910 PRINT:PRINT:PRINT
915 PRINT "PRESSIONE RETURN
PARA CONTINUAR"
```

```
920 GET X$
925 RETURN
950 HOME RETURN
960 LET E=PEEK (222)
965 IF E=8 THEN PRINT "*** ERRO
DE E/S" GOTO 995
970 IF E=6 THEN PRINT "*** ARQU
VO INEXISTENTE"  GOTO 995
970 IF E=10 THEN PRINT "*** ARQ
UIVO PROTEGIDO": GOTO 995
990 IF E=13 THEN PRINT "*** TIP
O ILEGAL DE ARQUIVO"·GOTO 995
992 IF E=11 THEN PRINT "*** ERR
O DE SINTAXE"
995 GOSUB 910:RUN
```

COMO FUNCIONA

As linhas 10 a 240 mostram o menu na tela. Ele varia conforme o modelo do computador, pois nem todas as suas funções são iguais, ou acessíveis por meio de um programa.

A linha 230 (**ON...GOTO**) desvia o programa para o trecho correspondente a cada função. O comando **ON ERR GOTO** logo na primeira linha serve para detectar erros de processamento de um comando do DOS. Caso isso aconteça, ele desvia o programa para o trecho em que será mostrada uma mensagem de erro (linha 960 em diante).

Após acionar a função do DOS que foi solicitada, o programa se auto-executa novamente, mostrando a tela de opções. A única exceção, em todos os microcomputadores, acontece quando se solicita a execução de um outro programa: não há retorno automático ao menu inicial, e você deve executá-lo mais uma vez, se quiser.

As linhas 910 e 950 marcam o início de duas sub-rotinas usadas repetidamente ao longo do programa. A primeira (910) serve para colocar uma mensagem no vídeo, avisando que o usuário deve pressionar a tecla <**RETURN**> ou <**ENTER**> para limpar a tela e voltar ao menu principal (nos micros das linhas MSX, TRS-80 e TRS-Color, pode-se pressionar qualquer tecla).

A sub-rotina na linha 950, por sua vez, é usada apenas para limpar a tela; ela foi colocada aí para facilitar a conversão do programa para outros micros. Assim, em vez de ter de alterar todas as linhas do programa onde ocorre um **CLS**, ou um **HOME** (e são muitas), é mais fácil modificar apenas uma.



Os micros da linha TRS-Color não têm um sistema operacional separado, como é o caso dos modelos das linhas TRS-80 e MSX. Ao se inserir o conector do acionador de disquetes na porta de cartuchos da máquina, o BASIC armazenado na ROM do computador fica apto para usar os comandos exclusivos para a operação do disquete. Esses comandos são poucos, em contraste, mais uma vez, com o TRS-80. As instruções principais — de apagamento (**KILL**), mudança de nome (**NAME**) e execução de programas em BASIC, diretamente do disco (**RUN**) —, contudo, são idênticas às do TRS-80; por isso os programas para ambas as linhas são similares. A única diferença diz respeito ao comando usado para listar o catálogo do disquete (**DIR**).



Os micros da linha TRS-80 contam com um sistema operacional separado, que pode ser invocado fora do BASIC. Esse sistema é armazenado nas trilhas do meio do disquete, e é carregado automaticamente toda vez que o computador é inicializado. Entretanto, o BASIC de disco tem alguns comandos semelhantes, em operação, aos comandos do DOS. Embora em pequeno número, esses comandos são a base do funcionamento do programa aqui apresentado. Os principais são os de apagamento (**KILL**), mudança de nome (**NAME**) e execução de programas em BASIC, diretamente do disco (**RUN**). Eles são idênticos aos do TRS-Color; por isso os programas para ambas as linhas são similares. A única diferença diz respeito à instrução usada para listar o catálogo do disquete (**DIR**) no TRS-Color. No TRS-80 é usado o comando genérico **CMD**, que dá acesso direto, a partir do BASIC, a várias funções do DOS. Para obter o catálogo, por exemplo, usa-se **CMD "D:0"** ou **CMD "D:1"**, dependendo do acionador em que está o disquete. Note que o número do acionador é perguntado pelo programa.



Como no caso anterior, os micros da linha MSX têm um sistema operacional separado, que pode ser invocado fora do BASIC. Armazenado nas trilhas iniciais do disquete, esse sistema é carregado automaticamente toda vez que o computador é inicializado. Entretanto, o BASIC de disco conta com algumas instruções semelhantes, em operação, aos comandos do DOS. Essas instruções formam a base do funcionamento do programa aqui apresentado. Os comandos principais são os de apagamento (**KILL**), mudança de nome (**NAME**) e execução de programas em BASIC, diretamente do disco (**RUN**). Eles são idênticos aos do TRS-80 e do TRS-Color; por isso os programas para ambas as linhas são similares. A única diferença refere-se ao comando que lista o catálogo do disquete (**DIR** no TRS-Color, **CMD** no TRS-80). No MSX, esse comando se chama **FILES**, e, em operação, é equivalente à instrução **DIR** do seu sistema operacional. Os acionadores de disquetes podem ser chamados de A ou B. Note que o programa pergunta o nome do acionador. O programa utiliza todos os comandos referidos.



Assim como os microcomputadores da linha TRS-Color, os modelos das linhas Apple e TK-2000 não dispõem de um sistema operacional de discos separado do BASIC. Ao ser utilizado com disquete, o interpretador de ambas as máquinas passa automaticamente a ter acesso aos comandos específicos necessários à sua operação.

Entretanto, há um pequeno problema: os comandos básicos, como **RUN**, **DELETE**, **RENAME**, **LOCK**, **UNLOCK**, **VERIFY** e **CATALOG**, só respondem quando usados de modo direto (isto é, fora de um programa). Para executá-los a partir de um programa, é necessário colocar a cadeia de comandos dentro de um **PRINT**, o qual deve transmitir antes o caractere ASCII 4; só agindo assim a manobra surtirá efeito.

Isso é explorado pelo programa, que, aliás, é o mais extenso de todos, em virtude da maior flexibilidade do Apple para explorar comandos do DOS a partir de um programa em BASIC.

LIMITAÇÕES

O programa apresentado atinge seu principal objetivo, pois, de fato, permite ao usuário operar com disquetes sem conhecer os comandos adicionais específicos. Porém, como todo programa que tenta substituir uma linguagem de comandos por um sistema de escolha por cardápios, o nosso "assistente" tem algumas limitações. Em outras palavras, da forma como está, ele não é capaz de explorar todas as possibilidades de uso dos comandos do DOS. Por exemplo, nos micros da linha MSX, ele não usa os caracteres ? e * para listar os arquivos de um determinado tipo, presentes no catálogo.

Se você conhece bem os comandos do DOS do seu micro, não será difícil modificar o programa apresentado, ampliando suas funções e utilidade.

RISCOS E PRÊMIOS AVALANCHE

Precisamos de incentivos e prêmios — como bolo e limonada — para ir em frente em nossa aventura. Mas não relaxe: as serpentes assassinas estão sempre à espera de um descuido.

Depois de montado, o cenário requer apenas algumas modificações para se adaptar aos diferentes níveis de dificuldade. No segundo nível, o grande problema de Willie consiste nos buracos existentes ao longo da encosta. Para desenhá-los, colocam-se blocos gráficos da cor do céu sobre o perfil da montanha. As cobras surgem apenas a partir do terceiro nível de dificuldade. A rotina que as movimenta será incorporada ao programa mais tarde.



```
 10 REM org 58455
 20 REM elb ld a,(57344)
 30 REM ld hl,57272
 40 REM ld b,a
 50 REM inc b
 60 REM ld de,8
 70 REM ab add hl,de
 80 REM djnz ab
 90 REM push hl
100 REM pop bc
110 REM ld hl,191
120 REM ld a,58
130 REM call print
140 REM ld a,(57344)
150 REM cp 0
160 REM jr z,ed
170 REM push af
180 REM call hls
190 REM pop af
200 REM cp 1
210 REM jr z,ed
220 REM call snp
230 REM ed ret
240 REM hls ld hl,457
250 REM call hlp
260 REM ld hl,401
270 REM call hlp
280 REM  ld hl,314
290 REM call hlp
300 REM ret
310 REM hlp ld b,4
320 REM hlq push bc
330 REM ld bc,15616
340 REM ld a,45
```



```
350 REM call print
360 REM ld de,32
370 REM add hl,de
380 REM pop bc
390 REM djnz hlq
400 REM ret
410 REM snp ld hl,457
420 REM call snq
430 REM ld hl,401
440 REM call snq
450 REM ld hl,314
460 REM call snq
470 REM ret
480 REM snq ld a,4
490 REM ld bc,57232
500 REM snr push af
510 REM ld a,43
520 REM call print
530 REM ld de,32
540 REM add hl,de
550 REM pop af
560 REM dec a
570 REM jr nz,snr
580 REM ret
590 REM print org 58217
```

OS PRÊMIOS

O conteúdo do endereço 57344 é colocado no acumulador. Esta posição de memória será utilizada para armazenar o nível de dificuldade em vigor. O par de registros HL recebe a seguir o endereço inicial dos padrões dos prêmios na tabela de dados. O nível de dificuldade é transferido do acumulador para o registro B, aumentando, então, em uma unidade. O par DE recebe o número 8 e é somado ao par HL.

O laço rotulado com **ad** prossegue adicionando 8 ao apontador HL até que o conteúdo de B se torne zero. Lembre-se de que a instrução **djnz** opera sobre o registro B: subtrai uma unidade e salta se o conteúdo deste não tiver chegado a zero. Tal processo movimenta o apontador do padrão do prêmio — o par de registro HL — pela tabela, até que ele aponte para o início do bloco gráfico adequado ao nível de dificuldade vigente. Cada bloco é um quadrado de oito por oito pontos, ocupando, assim, oito bytes na tabela.

O resultado dessas adições repetidas fica no par de registros HL. Contudo, a sub-rotina **print** precisa dele no par BC para imprimir o bloco correspondente. A maneira mais fácil e rápida de promover a transferência consiste em guardar o valor de HL na pilha, recuperando-o em seguida em BC.

A cor do caractere do prêmio é codificada pelo número 58, colocado em A. Depois disso, o programa chama a subrotina **print**, que desenha o bloco na cor apropriada. Essa rotina foi fornecida no primeiro artigo da série *Avalanche* (página 748).

O NÍVEL DE DIFICULDADE

O conteúdo da posição de memória usada para guardar o nível de dificuldade volta para o acumulador. O computador verifica inicialmente se ele é igual a zero. Em caso afirmativo, a instrução **jr z,ed** salta para o final da rotina, e a instrução **ret** provoca o retorno da mesma.

Se o nível vigente for diferente de zero, o processador preserva seu número, guardando-o na pilha. A sub-rotina **hls**, que desenha os buracos ao longo da encosta, é chamada a seguir. O nível de dificuldade é então recuperado da pilha e comparado a 1. Nesse nível, temos vários buracos mas não existe nenhuma serpente dentro deles. Assim, se A contiver 1, a instrução **cp 1** resulta em zero, o que ativa o indicador de zeros. A instrução **jr z,ed** salta, então, para o final da rotina, onde **ret** provoca o retorno.

Se o nível for maior que 1, a subrotina **snp**, que desenha as serpentes assassinas, é chamada. Depois disso, o programa termina.

Quando a rotina principal de controle estiver na memória, o processador retornará a ela. Por enquanto, ele retorna ao BASIC.

OS BURACOS

A sub-rotina seguinte tem a função de cavar buracos ao longo da encosta. Ela começa com o rótulo **hls**, que é chamado pela rotina principal quando é preciso acrescentar os buracos ao cenário. Três pequenos módulos, com duas instruções cada um, compõem essa subrotina. Eles começam com comandos do tipo **ld hl**, que colocam no par HL o endereço da tela correspondente à posição de impressão do topo de cada buraco. Após cada instrução desse tipo, a rotina **hlp** é chamada para colocar os blocos que formam o buraco sobre o desenho da encosta, feito pelos programas dados no último artigo.

A sub-rotina contém três módulos porque deve desenhar três buracos, cada um em uma posição diferente colocada em HL. Todos eles são feitos da mesma maneira, pela sub-rotina **hlp**.

MAIS BURACOS

Como cada buraco tem quatro blocos de profundidade, o número 4 é colocado no registro B e guardado temporariamente na pilha. O par BC recebe, então, o endereço de um espaço em branco. Na realidade, este é um byte da tabela de códigos para o alto da tela.

O código da cor do céu — 45 — vai para o acumulador e a sub-rotina **print** é novamente chamada. Assim, para colocar o primeiro bloco cor do céu sobre o perfil da encosta, o computador tira a primeira "pá de terra" do local onde ficará o buraco.

Colocando 32 em DE e somando esse valor a HL, fazemos com que este par aponte para o endereço da posição imediatamente inferior. O contador é recuperado da pilha e a instrução **djnz** subtrai uma unidade dele, saltando em direção ao rótulo **hlq**, se B ainda não for



zero. O processador executa esse laço quatro vezes, imprimindo mais um bloco cor do céu — ou tirando mais uma pá de terra — a cada volta.

AS COBRAS

O resto do programa desenha as cobras. As sete primeiras instruções são parecidas com as que iniciam a rotina dos buracos. Elas colocam no par HL a posição de impressão de cada cobra, chamando em seguida a sub-rotina que desenha a serpente.

Essa rotina também se assemelha àquela que desenha os buracos — afinal, as cobras devem caber dentro deles.

Desta vez, o contador ficará no acumulador, pois precisaremos controlar o apontador da tabela de padrões. No caso dos buracos, esse apontador foi direcionado para o endereço de um espaço em branco. Aqui ele percorrerá a tabela que criamos na memória RAM. A sub-rotina **print** atualiza automaticamente o apontador, movendo-o para o próximo byte. Como o apontador fica no par de registros BC, teríamos que ficar guardando seu valor na pilha, se quiséssemos recorrer à instrução **djnz** para controlar o laço. Com o contador em A, essa instrução é substituída por **dec a** e **jr nz, snr**.

TESTE DA ROTINA

Para testar a rotina, coloque níveis de dificuldade de 0 a 3 no endereço 57344, usando o comando **POKE 57344.** Chame a rotina com **RANDOM 58455** e verifique se os buracos estão sendo desenhados no nível 1; as cobras, nos níveis 2 e 3, e se prêmios diferentes aparecem a cada nível.



O programa listado a seguir desenha os buracos e as serpentes assassinas ao longo da encosta, conforme o nível de dificuldade.

```
10   ORG 19289
20   ELB LDA 18238
30   LDB #16
40   MUL
50   ADDD #18142
60   TFR D,U
70   LDX #2782
80   JSR CHARPR
90   LDA 18238
100   BEQ ED
110   PSHS A
120   JSR HOLES
130   PULS A
140   CMPA #1
150   BEQ ED
160   JSR SNAKE
170   ED RTS
180   HOLES LDX #5095
190   LDU #3071
200   JSR HOLPR
210   LDX #4591
220   LDU #3071
230   JSR HOLPR
240   LDX #3833
250   LDU #3071
260   JSR HOLPR
270   RTS
280   SNAKE LDX #5095
290   LDU #18078
300   JSR HOLPR
310   LDX #4591
320   LDU #18078
330   JSR HOLPR
340   LDX #3833
350   LDU #18078
360   JSR HOLPR
370   RTS
380   HOLPR LDB #4
390   HOLPRI PSHS U,X,B
400   JSR CHARPR
410   PULS B,X,U
420   LEAX 256,X
430   LEAU 16,U
440   DECB
450   BNE HOLPRI
460   [illegible]
470   CHARPR LDB #2
480   CHARI PSHS B,X
490   LDB #8
500   CHARZ PULU A
510   STA ,X
```

```
520  LEAX 32,X
530  DECB
540  BNE CHARZ
550  PULS X,B
560  LEAX 1,X
570  DECB
580  BNE CHARI
590  RTS
```

COMO TESTAR O PROGRAMA

Esse programa acrescenta blocos gráficos ao cenário já montado. Portanto, para testá-lo, é preciso que as rotinas e tabelas criadas anteriormente estejam na memória do micro. Depois de carregar os códigos e montar o programa, digite as linhas BASIC que seguem. Elas permitirão que você execute a parte do jogo já programada.

```
5 PCLEAR4:CLEAR 200,16999
10 EXEC 19000:EXEC 19109
20 POKE 18238,0
30 EXEC 19289
40 GOTO 40
```

A linha 10 executa parte do jogo montada nos artigos anteriores; a 20 acerta o valor do nível de dificuldade e a 30 executa o programa deste artigo. A 40 serve apenas para manter a tela de alta resolução ligada.

Para termos certeza de que nossa rotina funciona perfeitamente, devemos interromper o programa com **BREAK** e editar a linha 20 para mudar o nível de dificuldade. O nível 0 deixa o cenário intacto, apenas com a montanha — os perigos desse nível são as pedras, que faremos rolar morro abaixo em outra parte do programa.

O nível 1 desenha os buracos na encosta e o 2 acrescenta as serpentes assassinas. O nível 3 soma a tudo isso a avalanche, que, como dissemos, será promovida por outra rotina.

Teste cada um dos níveis. Verifique se as guloseimas roubadas estão sendo desenhadas no topo da montanha.

NÍVEIS DE DIFICULDADE

O endereço 18328 será usado para guardar o nível de dificuldade em vigor. Seu conteúdo transfere-se para o acumulador no início do programa e, em seguida, o número 16 é colocado no registro B. A instrução **MUL** multiplica o conteúdo desses dois registros, colocando o resultado em D.

Como o bloco gráfico de cada prêmio tem dezesseis bytes de comprimento, precisamos multiplicar o nível vigente por dezesseis, para obtermos na tabela o padrão do bloco a ele adequado.

O número obtido pela soma de 18142 ao resultado da multiplicação indica a posição inicial do padrão do bloco desejado, na tabela de padrões. Esse apontador é então transferido para o registro U, que funciona como apontador da pilha do usuário.

O registro X recebe o valor 2814, posição de impressão do prêmio na tela gráfica. Em seguida, o processador salta para o rótulo **CHARPR**, que imprime o bloco gráfico correspondente.

A instrução **LDA 18238** repõe o nível de dificuldade dentro do acumulador. Se o nível for zero, a instrução **BEQ** provoca um desvio para o rótulo **ED**. Como neste nível já nada mais precisa ser acrescentado ao cenário — com exceção do prêmio —, a instrução **RTS** termina a sub-rotina.

Se o nível for maior que zero, o programa guarda seu número na pilha da máquina. A sub-rotina que cava os buracos na encosta é, então, chamada.

Depois que a escavação termina, o nível é recuperado da pilha e comparado a 1. Se este for o número do nível atual, a instrução que desenha as cobras, **JSR SNAKE**, é ignorada e o processador retorna da sub-rotina.

BURACOS E COBRAS

As rotinas que desenham os buracos e as cobras compõem-se de três séries de três instruções, e funcionam da mesma maneira. Em cada série, o registro X recebe a posição da tela onde deve ser impresso o topo do bloco gráfico, enquanto o registro U recebe o endereço inicial do padrão do bloco na tabela. Em seguida, o processador salta para a rotina de impressão, **HOLPR**.

Como todos os buracos são iguais, o apontador da tabela tem sempre o valor inicial 3071. Esse endereço não fica na tabela, e sim na tela, pois o buraco nada mais é que um bloco cor do céu. O mesmo acontece com as cobras, cujo bloco gráfico tem endereço inicial 18078 na tabela de padrões.

As cobras são impressas, evidentemente, na mesma posição que os buracos que ocupam. O topo do primeiro buraco fica em 5095; o segundo, em 4591 e o terceiro, em 3833. Tanto as cobras quanto os buracos são desenhados sobre o perfil da montanha original. O restante do cenário permanece intacto.

A ROTINA DE IMPRESSÃO

As rotinas **HOLES** e **SNAKES** chamam a sub-rotina **HOLPR**, que é res-



ponsável tanto pelo desenho dos buracos quanto pelo das cobras.

O registro B recebe inicialmente o número 4 — os buracos têm quatro blocos de profundidade. Os registros U, X e B são guardados na pilha antes que a rotina **CHARPR** seja chamada. Esta é responsável pela impressão do bloco indicado pelo apontador de tabela de padrões na posição determinada por X.

Assim, se estivermos no nível 1 e formos desenhar um buraco, o apontador da tabela, U, indicará seu padrão e a rotina se encarregará de colocá-lo na posição dada por X. Se estivermos no nível 2 ou 3, U apontará para o padrão do bloco da cobra, que será impresso na posição X da tela.

Uma vez feita a impressão B, X e U são recuperados da pilha. X recebe então o valor X + 256, que corresponde à posição de impressão imediatamente inferior. U recebe o valor U + 16, que aponta para o padrão do próximo bloco gráfico na tabela.

Em seguida, B é subtraído em uma unidade e, se ainda não foi reduzido a zero, o processador volta ao rótulo **HOLPRI**. Quando o conteúdo desse registro chegar a zero, todos os quatro

blocos terão sido desenhados, a instrução **BNE HOLPRI** será ignorada e o processador retornará da sub-rotina.

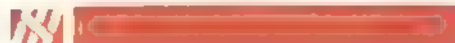
A impressão do bloco é feita pela rotina **CHARPR**, que explicamos a seguir.

Cada bloco gráfico impresso na tela tem dezesseis bits — dois bytes, portanto — de largura e oito bits de profundidade. O registro B recebe, assim, o valor 2, para contar a largura do buraco. Esse contador é preservado na pilha, juntamente com a posição de impressão contida no registro X. B recebe então o valor 8, para contar a profundidade do bloco.

O byte correspondente ao padrão da linha do bloco é recuperado da pilha do usuário, colocado no acumulador e, em seguida, transferido para a posição de impressão apontada pelo registro X. Por meio desse processo, o bloco é impresso linha por linha na tela.

A instrução **LEAX 32,X** coloca o valor X+32 no registro X, movendo o apontador de posições de tela uma linha de pixels para baixo. O contador B é diminuído em uma unidade e o processador retorna para imprimir o próximo byte, até que B tenha sido reduzido a zero — **BNE CHARZ**.

Em seguida, X e B são recuperados da pilha e X recebe o valor X+1 — **LEAX 1,X** —, o que faz o apontador mover-se um byte para a direita na tela. O contador B é então subtraído de uma unidade. Se ele ainda não tiver sido reduzido a zero, o processador retorna ao rótulo **CHARI** para imprimir mais uma coluna de oito bytes. Quando B já for igual a zero, as duas colunas estarão desenhadas e o processador retornará da sub-rotina **CHARPR**.



```
10    org 53682
20    ld a,(-5228)
30    rla
40    rla
50    ld b,56
60    add a,b
70    ld hl,(62407)
80    ld de,254
90    add hl,de
100    call 77
110    ld a,(-5228)
120    cp 0
130    jr z,ed
140    push af
150    call ho
160    pop af
170    cp 1
180    jr z,ed
190    call sn
200    ed ret
210    ho ld de,457
220    push de
230    ld de,401
240    push de
250    ld de,314
260    push de
270    ld c,3
280    po ld hl,(62407)
290    pop de
300    ld b,4
310    pp add hl,de
320    ld a,254
330    push hl
340    push bc
350    call 77
360    pop bc
370    pop hl
380    ld de,32
390    djnz pp
400    dec c
410    jr nz,po
420    ret
430    sn ld a,37
440    ld (-5227),a
450    add a,3
460    ld (-5226),a
470    add a,1
480    ld (-5225),a
490    add a,3
500    ld (-5224),a
510    ld de,457
520    push de
530    ld de,401
540    push de
550    ld de,314
560    push de
570    ld c,3
580    po ld hl,(62407)
590    ld de,-5227
600    ld (-5220),de
610    pop de
620    ld b,4
630    pp add hl,de
640    ld de,(-5220)
650    ld a,(de)
660    inc de
670    ld (-5220),de
680    push hl
690    push bc
700    call 77
710    pop bc
720    pop hl
730    ld de,32
740    djnz pp
750    dec c
760    jr nz,po
770    ret
780    end
```

ROTINA DE PRÊMIOS

O conteúdo do endereço -5228 é colocado no acumulador. O nível de dificuldade do jogo ficará armazenado nessa posição de memória.

O valor de A (0, 1, 2 ou 3) será utilizado para gerar o código da tabela de padrões que corresponde ao prêmio no nível vigente, ou seja:

nível	valor de A	prêmio	código
1	0	sanduíche	56
2	1	maçã	60
3	2	garrafa	64
4	3	sorvete	68

Armazenamos o código de padrão do sanduíche, 56, no registro B. Este será, portanto, o código-base. Em seguida, multiplicamos o conteúdo de A por quatro — a operação equivale a deslocar o conteúdo de A dois bits para a esquerda, o que é feito pela instrução **rla** executada duas vezes.

Somando o código-base contido em B ao valor de A, obtemos no acumulador o código do prêmio certo.

O endereço inicial da tabela de nomes de VRAM, que está nas posições de memória 62407 e 62408, é colocado no par de registros HL. Adicionamos a esse endereço o deslocamento correspondente à posição na qual desejamos imprimir o prêmio — 256, que equivale ao topo da montanha.

Para a impressão do bloco correspondente ao prêmio, utiliza-se a sub-rotina 77 da memória ROM. Esta coloca o valor do acumulador A na posição da VRAM apontada pelo par HL. No nosso caso; ela coloca o código do bloco do prêmio na tabela de nomes.

Lembre-se de que a tela, no modo de

alta resolução, é um reflexo da tabela de nomes da VRAM.

O NÍVEL DE DIFICULDADE

O conteúdo da posição de memória que guarda o nível de dificuldade é posto de volta no acumulador. Se for igual a zero, a instrução **jr z,ed** salta para o final da rotina onde se encontra a instrução **ret**. Caso contrário, o processador continua preservando o número do nível na pilha. A sub-rotina **hls**, que desenha os buracos ao longo da encosta, é então chamada.

O nível de dificuldade é recuperado da pilha e comparado a 1. Nesse nível temos vários buracos, mas nenhuma serpente dentro deles. Logo, se A contiver 1, a instrução **cp 1** resulta em zero, o que ativa o indicador de zeros. A instrução **jr z,ed** salta novamente para o final da rotina e retorna.

Se o nível for superior a 1, a subrotina **snp**, que desenha as serpentes assassinas, é chamada e, depois, o programa termina. Quando a rotina principal de controle estiver na memória, o programa retornará a ela. Por enquanto, retorna ao BASIC.

OS BURACOS

A próxima sub-rotina é totalmente dedicada a cavar buracos ao longo da encosta. Ela começa com o rótulo **ho**, chamado pela rotina principal quando surge a necessidade de acrescentar os buracos ao cenário. Inicialmente, **ho** armazena na pilha o endereço da tela correspondente à posição de impressão do topo de cada buraco. Como são três buracos, isso é feito três vezes.

Todos são desenhados da mesma maneira, por intermédio do laço **ho** que coloca no par de registros HL o endereço inicial da tabela de nomes da VRAM, adicionando a ele o endereço da tela correspondente a cada buraco.

O registro C é usado como contador para os três buracos, e a instrução **jr nz, ho** testa o fim da rotina.

Como cada buraco tem quatro blocos de profundidade, esse valor é colocado no registro B, que contém o número de execuções do laço **hp**.

O código da tabela de padrões do bloco cor de céu é armazenado no acumulador A. Em seguida, a rotina 77 da ROM, cujo funcionamento explicamos anteriormente, é chamada.

Para evitar possíveis alterações decorrentes da rotina 77, as instruções **pusch** e **pop** são usadas para proteger os contadores no par BC e os endereços no par HL. O contador é recuperado da pilha e a instrução **djnz** subtrai uma unidade dele, saltando em direção ao rótulo **hp** se o registro B ainda não tiver sido reduzido a zero.

O laço, executado quatro vezes, tira uma "pá de terra" a cada volta.

AS COBRAS ASSASSINAS

Cabe à rotina **sn** desenhar as cobras. O processo é basicamente igual ao da rotina **ho** e as cobras são desenhadas nas mesmas posições que os buracos. Há uma diferença fundamental: para cavar um buraco, precisamos colocar o mesmo padrão do bloco cor de céu quatro vezes; já para desenhar uma cobra temos que mudar o código do padrão para cada parte do animal.

Essa tarefa seria muito simples se os códigos dos padrões que formam a cobra estivessem em sequência. Porém, como você pode verificar na tabela de padrões, isso não ocorre. Armazenamos, assim, no início da rotina, os códigos dos padrões que formam a cobra — 37, 40, 41, 44 — nos endereços de -5227 até -5224. Depois, basta assegurar que esses endereços sejam chamados sequencialmente para cada cobra a ser desenhada. Isso é feito por intermédio do par DE, que serve como ponte, armazenando o endereço inicial da tabela (-5227) nas posições -5220 e -5219. Esses endereços contêm o endereço do código do padrão correspondente a cada uma das partes da cobra.

No mais, esta rotina é idêntica à que cava buracos.

TESTANDO

Para testar a rotina, digite o seguinte programa BASIC:

```
10 DEFUSR1=-12144
20 DEFUSR2=-11973
30 DEFUSR3=-11854
40 A=USR1(0)
50 A=USR2(0)
60 A=USR3(0)
70 GOTO 70
80 END
```

A linha 40 chama a rotina -12144, que coloca a tela no modo de alta resolução, e a 50 chama a rotina -11973, encarregada de desenhar a montanha na tela. A linha 60 chama a rotina apresentada neste artigo.

A tela é mantida no modo de alta resolução pela linha 70.

Para testar o funcionamento da rotina nos vários níveis de dificuldade, digite **POKE -5228,A**, atribuindo à variável A os valores 0, 1, 2 ou 3.

OS SEGREDOS DO TRS-80 (2)

- CÓPIA DA TELA
- FUNÇÃO **VARPTR** E VÍDEO
- TRANSFERÊNCIA
- APLICAÇÕES
- IMPRIMINDO A TELA

Pode-se consultar, com o BASIC, as posições da memória que armazenam o vídeo e as cadeias de caracteres. Usando esse recurso, você tornará seus programas bem mais rápidos.

No artigo anterior desta série, vimos como manipular o conteúdo da memória de vídeo do TRS-80 e compatíveis com o auxílio dos comandos **PEEK** e **POKE**. Podemos fazer isso porque o vídeo do TRS-80 é mapeado diretamente em um segmento fixo da RAM, que começa na locação 15360 e se estende até a locação 16376. Cada caractere na tela é armazenado em uma posição correspondente dessa memória de 1024 bytes.

Porém, os comandos **PEEK** e **POKE** são muito lentos quando usados dentro de laços **FOR...NEXT** para a transferência, consulta ou escrita de grande quantidade de caracteres na tela. Há uma técnica bem mais poderosa e rápida, que aprenderemos neste artigo: é o uso da função **VARPTR** como apontador de cadeias de caracteres para a tela.

Essa técnica permite armazenar instantaneamente 255 caracteres em exibição na tela, em uma cadeia alfanumérica (*string*). O limite de 255 caracteres (que corresponde a quatro linhas de vídeo) é imposto pelo tamanho máximo de um *string* alfanumérico. Se quisermos deslocar a tela inteira, portanto, basta efetuar a transferência em "partes" de 255 bytes de cada vez.

Para entender como a técnica funciona, precisamos conhecer o processo de armazenagem de cadeias alfanuméricas na memória do TRS-80. Existe um espaço na memória RAM, criado toda vez que o BASIC é inicializado. O argumento do comando **CLEAR** define o tamanho desse espaço (o tamanho predefinido é de cinquenta bytes).

Sempre que uma variável alfanumérica é definida (ou seja, sempre que não é uma constante dentro de um programa), o interpretador BASIC cria um par de bytes *apontadores*, que contém o endereço absoluto inicial da cadeia, na memória de *strings*. Nessa locação inicial, está armazenado um byte, que contém o comprimento da cadeia em bytes (daí a limitação intrínseca de 255 bytes por cadeia). Nas duas locações seguintes, encontram-se os bytes (o menos significativo e o mais significativo) do endereço inicial onde está armazenado o conteúdo da cadeia alfanumérica.

Seu programa pode ter acesso direto a toda essa informação através da função **VARPTR**. Para uma variável **V$**, por exemplo, teríamos:

VARPTR(V$) = locação do apontador de **V$**
PEEK(VARPTR(V$)) = comprimento de **V$**
PEEK(VARPTR(V$)+1) = endereço de **V$**
PEEK(VARPTR(V$)+2) = endereço de **V$**

Nada mais fácil, portanto, do que transferir 255 bytes da tela para uma variável alfanumérica: basta alterar o endereço armazenado nas locações **VARPTR(V$)+1** e **VARPTR(V$)+2** de modo que passe a apontar para a locação inicial da memória de vídeo que queremos transferir. Nenhum byte do conteúdo da tela precisa ser movido!

A sub-rotina seguinte coloca o comprimento desejado e o endereço inicial de vídeo no bloco de apontadores de uma variável **V$**:

```
1000 'SUBROTINA VTRANSF
1010 V$=" "·POKE VARPTR(V$).C%
1020 POKE VARPTR(V$)+1. V% INT
(V%/256)*256
1030 POKE VARPTR(V$)+2. INT(V%
/256)+60
1040 RETURN
```

A linha 1010 define **V$** e armazena no primeiro endereço do bloco de apontadores o número de bytes de comprimento (a variável inteira **C%**).

As linhas 1020 e 1030 tratam de armazenar no bloco de apontadores, respectivamente, o byte mais significativo e o byte menos significativo do endereço desejado de início no vídeo (a variável inteira **V%**)

Observe que as variáveis **C%** e **V%** estão definidas como inteiras. Para usar a sub-rotina, deve-se colocar em **V%** o endereço da posição de início na tela (0 a 1023), e, em **C%**, o número de bytes a se transferir (0 a 255). Ao retornar da sub-rotina, **V$** conterá os caracteres que estavam armazenados na tela, da posição **V%** a **V%+C%**.

APLICAÇÕES

Essa rotina, que chamaremos de **VTRANSF**, possui numerosas aplicações. Eis aqui algumas delas:

- copiar o conteúdo de uma tela de vídeo na impressora;
- armazenar o conteúdo de várias telas de vídeo na memória;
- armazenar as telas em disco;
- realizar animações gráficas, pela superposição rápida de porções de telas de vídeo previamente armazenadas em memória;
- montar vídeos com várias "janelas" superpostas, independentes.

Nos próximos artigos desta série, trataremos em detalhe das aplicações da rotina **VTRANSF**.

COMO IMPRIMIR A TELA

O programa que se segue é um primeiro exemplo de utilização da sub-rotina **VTRANSF**. Ele imprime o conteúdo corrente da tela em uma impressora, linha por linha:

```
10 CLEAR 500
20 C%-64
30 FOR V%-0 TO 960 STEP 64
40 GOSUB 1000 : LPRINT VS
50 NEXT V%
60 END
```

O programa funciona da seguinte maneira: a linha 10 reserva quinhentos bytes de memória para *strings*. A linha 20 define o comprimento **C%** da variável alfa que receberá cópia de uma linha de vídeo. A linha 30 inicia um laço, que vai até a linha 50, percorrendo os endereços de vídeo (**V%**) desde a primeira até a última linha, de 64 em 64 posições. Finalmente, a linha 40 chama nossa sub-rotina **VTRANSF** (que deverá ser colocada logo após a linha 60) e imprime a variável **V$**, que recebeu a cópia do vídeo.

Se sua impressora não for do tipo gráfico, use esse programa apenas para copiar telas de texto, pois ele não converte os caracteres gráficos (129 a 191) que encontra na tela.

O JOGO A RAPOSA E OS GANSOS (3)

As rotinas deste artigo permitem ao computador não só jogar pela raposa e pelos gansos como também estudar os próximos movimentos para melhorar o nível da partida.

Terceira e última parte do jogo *A Raposa e os Gansos*, este artigo apresenta rotinas que possibilitam ao computador jogar tanto pela raposa quanto pelos gansos. Na verdade, ele pode ser colocado a jogar sozinho ou simplesmente pode não jogar, se você tiver um parceiro humano; neste caso, o computador servirá apenas como tabuleiro e fornecedor de peças.

```
210 LET P=B(G(1))+B(G(2))+B(G(
3))+B(G(4))
220 LET X=F: IF B<B(32) THEN
LET P=P-BX: LET X=33-F
230 LET P=P*B(X): RETURN
250 LET F=FN A(ABS P)-30: LET
B=P/B(F): IF B<0 THEN  LET B=B
+BX: LET F=33-F
260 FOR A=1 TO 4: LET G(A)=FN
A(B)+I· LET B=B-B(G(A)): NEXT
A· RETURN
```



```
210 P=B(G(1))+B(G(2))+B(G(3))+B
(G(4))
220 X=F:IF P<B(31) THEN P=P-BX:
X=31-F
230 P=P*B(X)·RETURN
250 F=FNA(ABS(P))-31:B=P/B(F):I
F B<0 THEN B=B+BX:F=31-F
260 FOR A=1 TO 4·G(A)=FNA(B):B=
B-B(G(A)):NEXT:RETURN
```



```
210 P=B(G(1))+B(G(2))+B(G(3))+B
(G(4))
220 X=F:IF P<B(31) THEN P=P-BX:
X=31-F
230 P=P*B(X):RETURN
250 F=FNA(ABS(P))-31:B=P/B(F)·I
F B<0 THEN B=B+BX:F=31-F
260 FOR A=1 TO 4·G(A)=FNA(B):B=
B-B(G(A)):NEXT:RETURN
```



```
210 P = B(G(1)) + B(G(2)) + B(G
(3)) + B(G(4))
220 X = F: IF P < B(31) THEN P
= P - BX:X = 31 - F
230 P.= P * B(X): RETURN
250 F =  FN A( ABS (P)) - 31:B
= P / B(F): IF B < 0 THEN B = B
 + BX:F = 31 - F
260  FOR A = 1 TO 4·G(A) =  FN
A(B):B = B - B(G(A)): NEXT : RE
TURN
```

Essas são duas das mais importantes rotinas do programa. A rotina da linha 210 à linha 230 avalia a posição em jogo e a transforma em um número. De modo inverso, quando o computador tiver escolhido a melhor jogada, tem que converter um número em um movimento. As linhas 250 a 260 transformam o valor numérico em um conjunto de valores necessários para posicionar as peças. Essas sub-rotinas são chamadas com muita freqüência e, por isso, foram colocadas logo no início do programa.

O MOVIMENTO DA RAPOSA



```
1010 GOSUB 210
1020 GOSUB 310· GOSUB 250
1030 IF F>28 THEN  PRINT AT 21.
0;"A RAPOSA VENCEU
     ": GOTO 1410
1032 GOSUB 410: IF V=H THEN  PR·
```

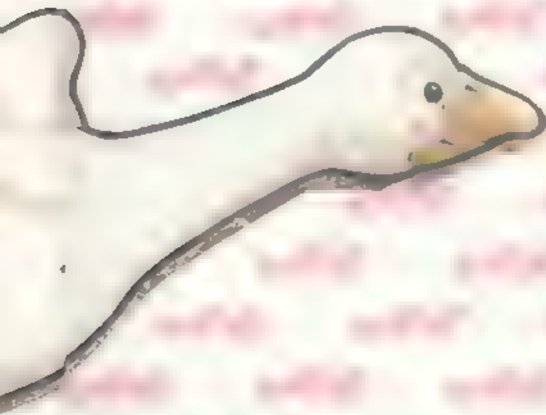


```
INT "OS GANSOS VENCERAM
     ": GOTO 1410
1040 IF PF THEN  GOTO 1110
1050 INPUT "MOVER RAPOSA PARA "
;B· IF B=-1 THEN  GOSUB 2710: G
OTO 1030
1055 GOSUB 4000
1060 FOR A=1 TO X(F)· LET X=M(A
,F): IF X=B THEN  IF NOT (FN X(
X)) THEN  LET F=B· GOSUB 210: G
OTO 1200
1070 NEXT A· PRINT AT 21.0:"ISS
O NAO E LEGAL                 ":
 GOTO 1050
1110 LET L=SF: LET M=SF: LET V(
M)=E*M: IF M>4 THEN  DIM R(HF+3
): DIM S(HF+3)
1112 GOSUB 1120: GOTO 1200
```



```
1120 IF L=1 THEN  GOTO 410
1122 IF L<M-2 THEN  GOSUB 1610:
 IF V<>0 THEN  RETURN
1130 LET L=L-1: LET V(L)=H*L: L
ET A(L)=X(F): LET F(L)=F
1140 LET F=M(A(L),F(L))
1150 IF FN X(F)=0 THEN  GOSUB 1
320: IF V>V(L) THEN  LET V(L)=V:
  LET P(L)=F: IF V>V(L+1) THEN
 LET F=F(L): LET L=L+1: RETURN
1160 LET A(L)=A(L)-1: IF A(L)>0
 THEN  GOTO 1140
1170 LET V=V(L): LET F=F(L): LE
T L=L+1: IF L=M THEN  LET F=P(M
-1): GOSUB 210: RETURN
1172 IF L<M-2 THEN  GOSUB 1510:
 RETURN
1180 RETURN
```







```
1010 SCREEN 1,0:GOSUB 210:GOTO
1030
1020 XX=FNXX(G):YY=FNYY(G):PUT(
XX,YY)-(XX+19,YY+19),SQ,PSET:XX
=FNXX(G(C)):YY=FNYY(G(C))+5:PUT
(XX,YY)-(XX+19,YY+9),GS,PSET
1030 CLS:IF F>27 THEN PLAY V$:P
RINT @7,"A RAPOSA VENCEU":GOTO
1410
1032 GOSUB 410:IF V=H THEN PLAY
 V$:PRINT @6,"OS GANSOS VENCERA
M":GOTO 1410
1040 LINE(180,0)-(255,191),PRES
ET,BF:IF PF THEN DRAW"BM180,50C
4"+TH$:GOTO 1110
1050 DRAW"BM180,80C4"+MW$:XX=FN
XX(F):YY=FNYY(F):GOSUB 1810:B=4
*INT(YY/20):B=FNCN(B)
1055 IF B=-1 THEN PLAY V$:PRINT
@6,"OS GANSOS VENCERAM":GOTO 14
10
1060 FOR A=0 TO X(F):X=M(A,F):I
F X=B AND NOT FNX(X) THEN A=X(F
):XX=FNXX(F):YY=FNYY(F):PUT(XX,
YY)-(XX+19,YY+19),SQ,PSET:F=B:G
OSUB 210:NEXT:GOTO 1200
1070 NEXT:GOSUB 5000:GOTO 1040
1110 L=SF:M=SF:V(M)=E*M:IF M>4
THEN FOR A=0 TO HF:R(A)=0:NEXT
1112 LF=F:GOSUB 1120:XX=FNXX(LF
):YY=FNYY(LF):PUT(XX,YY)-(XX+19
,YY+19),SQ,PSET:GOTO 1200
1120 IF L=1 THEN 410
1122 IF L<M-2 GOSUB 1610:IF V<>
0 THEN RETURN
1130 L=L-1:V(L)=H*L:A(L)=X(F):F
(L)=F
1140 F=M(A(L),F(L))
1150 IF FNX(F)=0 GOSUB 1320:IF
V>V(L) THEN V(L)=V:P(L)=F:IF V>
V(L+1) THEN F=F(L):L=L+1:RETURN
1160 A(L)=A(L)-1:IF A(L)>=0 THE
N 1140
1170 V=V(L):F=F(L):L=L+1:IF L=M
 THEN F=P(M-1):GOSUB 210:RETURN
1172 IF L<M-2 GOSUB 1510
1180 RETURN
```



```
1010 GOSUB 210:GOTO 1030
1020 GS(FNZZ(G),1)=G(C):XX=FNXX
(G(C)):YY=FNYY(G(C)):PUT SPRITE
 GS(FNZZ(G(C)),0),(XX,YY),15
1030 IF F>27 THEN PLAYV$:FORK=1
TO500:NEXT:SCREEN0:PRINT"A RAPO
SA VENCEU!":GOTO 1410
1032 GOSUB 410:IF V=H THEN PLAY
V$:FORK=1TO500:NEXT:SCREEN0:PRI
NT"OS GANSOS VENCERAM!":GOTO 14
10
1040 LINE(188,0)-(255,191),11,B
F:IF PF THEN PRESET (188,50):PR
INT#1,"Pensando":GOTO 1110
1050 PRESET (188,80):PRINT#1,"P
ara ?":XX=FNXX(F):YY=FNYY(F):GO
SUB 1810:B=4*INT((YY-2)/20):B=F
NCN(B)
1055 IF B=-1 THEN PLAYV$:FORK=1
TO500:NEXT:SCREEN0:PRINT"OS GAN
SOS VENCERAM!":GOTO 1410
1060 FOR A=0 TO X(F):X=M(A,F):I
F X=B AND NOT FNX(X) THEN A=X(F
):F=B:GOSUB 210:NEXT:GOTO 1200
1070 NEXT:GOSUB 5000:GOTO 1040
1110 L=SF:M=SF:V(M)=E*M:IF M>4
THEN FOR A=0 TO HF:R(A)=0:NEXT
1112 LF=F:GOSUB 1120:GOTO 1200
1120 IF L=1 THEN 410
1122 IF L<M-2 THEN GOSUB 1610:I
F V<>0 THEN RETURN
1130 L=L-1:V(L)=H*L:A(L)=X(F):F
(L)=F
1140 F=M(A(L),F(L))
1150 IF FNX(F)=0 THEN GOSUB 132
0:IF V>V(L) THEN V(L)=V:P(L)=F:
IF V>V(L+1) THEN F=F(L):L=L+1:R
ETURN
1160 A(L)=A(L)-1:IF A(L)>=0 THE
N 1140
1170 V=V(L):F=F(L):L=L+1:IF L=M
 THEN F=P(M-1):GOSUB 210:RETURN
1172 IF L<M-2 THEN GOSUB 1510
1180 RETURN
```





```
1010  GOSUB 210: GOTO 1030
1020 XX =  FN XX(G):YY =  FN YY
 (G): HCOLOR= 0: DRAW GS AT XX +
 7,YY + 5: HCOLOR= 3:XX =  FN X
X(G(C)):YY =  FN YY(G(C)): DRAW
 GS AT XX + 7,YY + 5
1030  HOME : IF F > 27 THEN  PR
INT V$: HOME : VTAB 22: PRINT "
A RAPOSA GANHOU!": GOTO 1410
1032  GOSUB 410: IF V = H THEN
 PRINT V$: HOME : VTAB 22: PRIN
T "OS GANSOS GANHARAM!": GOTO 1
410
1040  HOME : VTAB 22: IF PF THE
N  PRINT "PENSANDO...";: GOTO 1
110
1050  PRINT "MOVE PARA ONDE? ";
:XX =  FN XX(F):YY =  FN YY(F):
 GOSUB 1810:B = 4 *  INT (YY /
20):B =  FN CN(B)
1055  IF B = - 1 THEN  PRINT V
$: HOME : VTAB 22: PRINT "OS GA
NSOS VENCERAM!": GOTO 1410
1060  FOR A = 0 TO X(F):X = M(A
,F): IF X = B AND  NOT  FN X(X)
 THEN A = X(F):XX =  FN XX(F):Y
Y =  FN YY(F): HCOLOR= 0: DRAW
FX AT XX + 2,YY + 8:F = B: HCOL
OR= 3: GOSUB 210: NEXT : GOTO 1
200
1070  NEXT : GOSUB 5000: GOTO 1
040
1110 L = SF:M = SF:V(M) = E * M
: IF M > 4 THEN  FOR A = 0 TO H
F:R(A) = 0: NEXT
1112 LF = F: GOSUB 1120:XX =  F
N XX(LF):YY =  FN YY(LF): HCOLO
R= 0: DRAW FX AT XX + 2,YY + 8:
 HCOLOR= 3: GOTO 1200
1120  IF L = 1 THEN 410
1122  IF L < M - 2 THEN  GOSUB
1610: IF V <  > 0 THEN  RETURN

1130 L = L - 1:V(L) = H * L:A(L
) = X(F):F(L) = F
1140 F = M(A(L),F(L))
1150  IF  FN X(F) = 0 THEN  GOS
UB 1320: IF V > V(L) THEN V(L)
 = V:P(L) = F: IF V > V(L + 1) T
HEN F = F(L):L = L + 1: RETURN
1160 A(L) = A(L) - 1: IF A(L) >
 = 0 THEN 1140
1170 V = V(L):F = F(L):L = L +
1: IF L = M THEN F = P(M - 1):
GOSUB 210: RETURN
1172  IF L < M - 2 THEN  GOSUB
1510
1180  RETURN
```

As linhas 1020 a 1180 manipulam os movimentos da raposa. A sub-rotina que desenha o tabuleiro é usada para mostrar o posicionamento atual das peças. Em seguida, verifica-se na linha 1030 se a raposa ganhou. Feito isso, a rotina comprova na linha 1032 se há pelo menos um movimento legal para a raposa (caso contrário a vitória caberá aos gansos).

Se o jogador estiver controlando a raposa, as linhas 1050 a 1070 receberão a jogada e verificarão sua validade. Se, ao contrário, quem estiver controlando a raposa for o computador, as linhas 1110 a 1112 decidirão pelo movimento. O número de lances que está sendo analisado, **M**, e o lance que é analisado no momento, **L**, são definidos na linha 1110. As linhas 1120 a 1180 avaliam qual a melhor jogada.

Quatro matrizes são usadas nessa rotina: **A** contém o número de movimentos a serem analisados; **V**, o melhor resultado até o momento; **P**, a jogada que provoca tal resultado, e **F** indica a posição prévia da raposa.

COMO MOVER OS GANSOS



```
500 LET V=0: FOR C=1 TO 4: LET
G=G(C): FOR A=1 TO Z(G): LET X
=M(A,G): IF X<>F THEN  IF NOT
FN X(X) THEN  RETURN
502 NEXT A: NEXT C: LET V=1:
 RETURN
1200 GOSUB 310: GOSUB 250
1202 IF F>28 THEN  PRINT AT 21,
0;"A RAPOSA VENCEU
    ": GOTO 1410
1204 GOSUB 500  IF V THEN  PRIN
T AT 21,0,"A RAPOSA VENCEU
            ": GOTO 1410
1210 IF PG THEN  GOTO 1310
1220 INPUT "QUE GANSO Q ER MOVE
R ?";G: IF G=-1 THEN  GOSUB 271
0: GOTO 1202
1225 GOSUB 4000
1230 LET C=FN Z(G): IF C=0 THEN
  PRINT AT 21,0;"NAO HA GANSO E
M ",G,""  GOTO 1220
1240 INPUT "PARA ONDE ";I: IF I
=-1 THEN  GOSUB 2710  GOTO 1202
1250 IF FN X(I) THEN  PRINT AT
21,0;"AI JA TEM UM GANSO !
      ": GOTO 1220
1260 IF I=F THEN  PRINT AT 21,0
;"AI JA TEM UMA RAPOSA
```



```
  ": GOTO 1220
1270 FOR A=1 TO Z(G): IF M(A,G)
=I THEN  LET G(C)=I: GOTO 1010
1280 NEXT A: PRINT AT 21,0;"ISS
O E ILEGAL
 GOTO 1220
1310 LET L=SG: LET M=SG: LET V(
M)=H*M  IF M>4 THEN  DIM R(HF+3
): DIM S(HF+3)
1312 GOSUB 1320: GOTO 1020
1320 IF L=1 THEN  GOTO 510
1322 IF L<M-2 THEN  GOSUB 1610:
 IF V<>0 THEN  RETURN
1324 LET L=L-1: LET V(L)=E*L: L
ET C=1
1330 LET C(L)=C: LET F(L)=G(C)
 LET A(L)=1: IF A(L)>Z(G(C)) TH
EN  GOTO 1362
1340 LET B=M(A(L),F(L)): LET X=
FN X(B): LET G(C)=B: IF X OR B=
F THEN  GOTO 1360
1350 GOSUB 1120: LET C=C(L): IF
 V<V(L) THEN  LET V(L)=V: LET P
(L)=G(C)+C*32
1355 IF V<V(L) THEN  LET G(C)=F
```

```
(L): LET L=L+1· RETURN
1360 LET A(L)=A(L)+1: IF A(L)<=
Z(F(L)) THEN  GOTO 1340
1362 LET G(C)=F(L): LET C=C+1:
IF C<5 THEN  GOTO 1330
1370 LET V=V(L): LET L=L+1: IF
L=M THEN  LET C=INT (P(L-1)/32)
· LET G(C)=P(L-1)-C*32: GOSUB 2
10: RETURN
1372 IF L<M-2 THEN  GOSUB 1510·
RETURN
1380 RETURN
```



```
500 V=-1:FOR C=1 TO 4:G=G(C)·IF
 Z(G)<0 THEN NEXT:RETURN
502 FOR A=0 TO Z(G):X=M(A,G):V=
V AND(FNX(X) OR X=F):NEXT A,C:R
ETURN
1200 GOSUB 250:XX=FNXX(F):YY=FN
YY(F)+5:PUT(XX,YY)-(XX+19,YY+8)
,FX,PSET
```



```
1202 IF F>27 THEN PLAY V$:PRINT
 #6,"A RAPOSA VENCEU":GOTO 1410
1204 GOSUB 500·IF V THEN PLAY V
$ PRINT "A RAPOSA VENCEU":GOTO
1410
1210 LINE(180,0)-(255,191),PRES
ET,BF·IF PG THEN DRAW"BM180,50C
4"+TH$·GOTO 1310
1220 XX=FNXX(G(1))·YY=FNYY(G(1)
) DRAW"BM180,80C2"+WG$:GOSUB 18
10
1225 G=4*INT(YY/20)·G=FNCN(G)
1230 C=FNZ(G):IF C=0 GOSUB 5000
·GOTO 1210
1240 DRAW"BM180,110C3"+MW$:GOSU
B 1810:I=4*INT(YY/20):I=FNCN(I)
1245 IF I=1 THEN PLAY V$:PRINT
#7,"A RAPOSA VENCEU":GOTO 1410
1250 IF FNX(I) GOSUB 5000·GOTO
1210
1260 IF I=F GOSUB 5000:GOTO 121
0
1270 FOR A=-1 TO Z(G):IF A>=0 T
HEN IF M(A,G)=I THEN XX=FNXX(G(
C)):YY=FNYY(G(C)):PUT(XX,YY)-(X
X+19,YY+19),SQ,PSET:G(C)=I:XX=F
NXX(I):YY=FNYY(I):PUT(XX,YY+5)-
(XX+19,YY+14),GS,PSET:A=Z(G):NE
XT:GOTO 1010
1280 NEXT·GOSUB 5000·GOTO 1210
1310 L=SG:M=SG:V(M)=H*M:IF M>4
THEN FOR A=0 TO HF:R(A)=0:NEXT
1312 GOSUB 1320:GOTO 1020
1320 IF L=1 THEN 510
1322 IF L<M-2 GOSUB 1610:IF V<>
0 THEN RETURN
1324 L=L-1:V(L)=E*L:C=1
1330 C(L)=C:F(L)=G(C):A(L)=0:IF
 A(L)>Z(G(C)) THEN 1362
1340 B=M(A(L),F(L)):X=FNX(B):G=
G(C):G(C)=B:IF X OR B=F GOTO 13
60
1350 GOSUB 1120:C=C(L):IF V<V(L
) THEN V(L)=V:P(L)=G(C)+C*32:IF
 V<V(L+1) THEN G=G(C):G(C)=F(L)
:L=L+1·RETURN
1360 A(L)=A(L)+1:IF A(L)<=Z(F(L
)) THEN 1340
1362 G=G(C):G(C)=F(L):C=C+1:IF
C<5 THEN 1330
1370 V=V(L):L=L+1:IF L=M THEN C
=INT(P(L-1)/32):G=G(C):G(C)=P(L
-1) AND 31:GOSUB 210:RETURN
1372 IF L<M-2 GOSUB 1510
1380 RETURN
```



```
500 V=-1·FOR C=1 TO 4:G=G(C)·IF
 Z(G)<0 THEN NEXT·RETURN
502 FOR A=0 TO Z(G)·X=M(A,G)·V=
V AND (FNX(X) OR X=F):NEXT:NEXT
:RETURN
1200 GOSUB 250·XX=FNXX(F)·YY=FN
YY(F):PUT SPRITE FX,(XX,YY),6
1202 IF F>27 THEN PLAYV$:FORK=1
TO500·NEXT·SCREEN0·PRINT"A RAPO
SA VENCEU!":GOTO 1410
1204 GOSUB 500·IF V THEN PLAYV$
·FORK=1TO500·NEXT SCREEN0·PRINT
"A RAPOSA VENCEU!"·GOTO 1410
1210 LINE (188,0)-(255,191),11,
BF·IF PG THEN PRESET (188,50):P
RINT#1,"Pensando" GOTO 1310
1220 XX=FNXX(G(1)):YY=FNYY(G(1)
):PRESET (188,80):PRINT#1,"Qual
 ?":GOSUB 1810
1225 G=4*INT((YY-2)/20):G=FNCN(
G)
1230 C=FNZ(G):IF C=0 THEN GOSUB
 5000:GOTO 1210
1240 LINE (188,0)-(255,191),11,
BF:PRESET (188,80):PRINT#1,"Par
a ?":GOSUB 1810:I=4*INT((YY-2)/
20):I=FNCN(I)
1245 IF I=1 THEN PLAYV$:FORK=1T
O500:NEXT:SCREEN0:PRINT"A RAPOS
A VENCEU!":GOTO 1410
1250 IF FNX(I) THEN GOSUB 5000:
GOTO 1210
1260 IF I=F THEN GOSUB 5000·GOT
O 1210
1265 GS(FNZZ(G),1)=I
1270 FOR A=-1 TO Z(G):IF A>=0 T
HEN IF M(A,G)=I THEN XX=FNXX(I)
:YY=FNYY(I).PUT SPRITE GS(FNZZ(
I),0),(XX,YY),15·A=Z(G):G(C)=I:
NEXT·GOTO 1010
1280 NEXT:GOSUB 5000·GOTO 1210
1310 L=SG·M=SG·V(M)=H*M·IF M>4
THEN FOR A=0 TO HF R(A)=0·NEXT
1312 GOSUB 1320·GOTO 1020
1320 IF L=1 THEN 510
1322 IF L<M-2 THEN GOSUB 1610:I
F V<>0 THEN RETURN
1324 L=L-1·V(L)=E*L·C=1
1330 C(L)=C·F(L)=G(C)·A(L)=0·IF
 A(L)>Z(G(C)) THEN 1362
1340 B=M(A(L),F(L))·X=FNX(B)·G=
G(C):G(C)=B:IF X OR B=F THEN 13
60
1350 GOSUB 1120·C=C(L)·IF V<V(L
) THEN V(L)=V:P(L)=G(C)+C*32:IF
 V<V(L+1) THEN G=G(C):G(C)=F(L)
:L=L+1·RETURN
1360 A(L)=A(L)+1:IF A(L)<=Z(F(L
)) THEN 1340
1362 G=G(C)·G(C)=F(L)·C=C+1:IF
C<5 THEN 1330
1370 V=V(L):L=L+1:IF L=M THEN C
=INT(P(L-1)/32)·G=G(C)·G(C)=P(L
-1)AND31:GOSUB 210·RETURN
1372 IF L<M-2 THEN GOSUB 1510
1380 RETURN
```



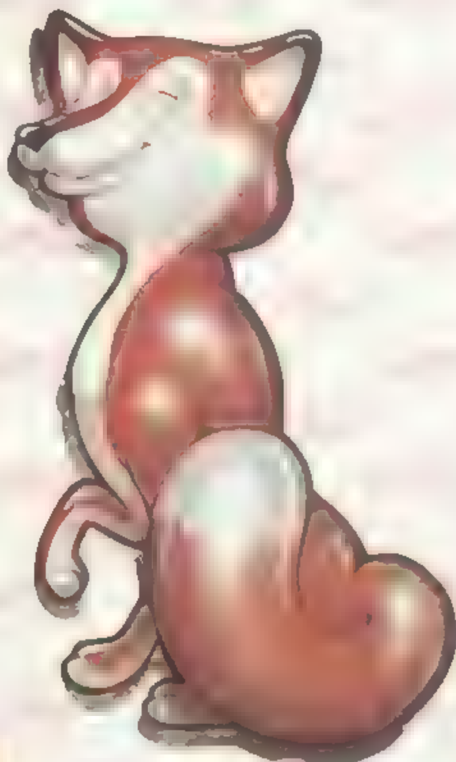


```
500 V = 1: FOR C = 1 TO 4:G = G
(C): IF G(C) < 0 THEN  NEXT : R
ETURN
502  FOR A = 0 TO Z(G):X = M(A,
G):V = V AND ( FN X(X) OR X = F
): NEXT : NEXT : RETURN


1200  GOSUB 250·XX =  FN XX(F):
YY =  FN YY(F): DRAW FX AT XX +
 2,YY + 8
1202  IF F > 27 THEN  PRINT V$·
 HOME · VTAB 22: PRINT "A RAPOS
A VENCEU!"· GOTO 1410
1204  GOSUB 500· IF V THEN  PRI
NT V$: HOME : VTAB 22: PRINT "A
 RAPOSA VENCEU!": GOTO 1410
1210  HOME · VTAB 22· IF PG THE
N  PRINT "PENSANDO...":: GOTO 1
310
1220 XX =  FN XX(G(1)):YY =  FN
 YY(G(1)): PRINT "MOVE QUAL GAN
SO?": GOSUB 1810
1225 G = 4 *  INT (YY / 20):G =
  FN CN(G)
1230 C =  FN Z(G): IF C = 0 THE
N  GOSUB 5000: GOTO 1210
1240  HOME : VTAB 22· PRINT "MO
VE PARA ONDE? ": GOSUB 1810:I =
 4 *  INT (YY / 20):I =  FN CN(
I)
1245  IF I = 1 THEN  PRINT VS:
HOME : VTAB 22: PRINT "A RAPOSA
 VENCEU!": GOTO 1410
1250  IF  FN X(I) THEN  GOSUB 5
000· GOTO 1210
1270  FOR A =  - 1 TO Z(G): IF
A >  = 0 THEN  IF M(A,G) = I TH
EN XX =  FN XX(G(C))·YY =  FN Y
Y(G(C)). HCOLOR= 0. DRAW GS AT
XX + 7,YY + 5:G(C) = I:XX =  FN
 XX(I)·YY =  FN YY(I)· HCOLOR=
3: DRAW GS AT XX + 7,YY + 5:A =
 Z(G): NEXT : GOTO 1010
1280  NEXT : GOSUB 5000· GOTO 1
210
1310 L = SG:M = SG:V(M) = H * M
: IF M > 4 THEN  FOR A = 0 TO H
F:R(A) = 0: NEXT
1312  GOSUB 1320: GOTO 1020
1320  IF L = 1 THEN 510
1322  IF L < M - 2 THEN  GOSUB
1610: IF V <  > 0 THEN  RETURN
1324 L = L - 1:V(L) = E * L:C =
 1
1330 C(L) = C:F(L) = G(C):A(L)
= 0: IF A(L) > Z(G(C)) THEN 136
2
1340 B = M(A(L),F(L)):X =  FN X
(B)·G = G(C):G(C) = B: IF X OR
B = F THEN 1360
1350  GOSUB 1120·C = C(L): IF V
 < V(L) THEN V(L) = V:P(L) = G(
C) + C * 32· IF V < V(L + 1) TH
EN G = G(C):G(C) = F(L):L = L +
 1: RETURN
1360 A(L) = A(L) + 1· IF A(L) <
 = Z(F(L)) THEN 1340
1362 G = G(C):G(C) = F(L):C = C
 + 1: IF C < 5 THEN 1330
1370 V = V(L)·L = L + 1· IF L =
 M THEN C =  INT (P(L - 1) / 32
):G = G(C):MD = 32:G(C) =  FN M
D(P(L - 1)): GOSUB 210: RETURN

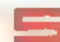
1372  IF L < M - 2 THEN  GOSUB
1510
1380  RETURN
```





A rotina que vai da linha 1200 à linha 1380 é responsável pela manipulação dos gansos. Uma parte dela — linhas 1220 a 1290 — é utilizada quando o jogador controla os gansos. Quando esse controle cabe ao microcomputador, as linhas 1320 a 1380 encarregam-se da manipulação.

MELHORES JOGADAS



```
410 LET V=H: FOR A=X(F) TO 1
STEP -1: LET X=M(A,F): IF FN X
(X) THEN  NEXT A: LET L=1:
RETURN
420 LET B=F: LET F=X: GOSUB
210: LET V=P: LET F=B: LET L=1
: RETURN
510 LET V=E: FOR C=1 TO 4: LET
G=G(C): IF -B(G)>V THEN  GOTO
 530
520 FOR A=1 TO Z(G): LET X=M(A
,G): IF FN X(X) OR (X=F) THEN
NEXT A: GOTO 530
528 LET V=B(X)-B(G): LET D=C:
LET B=X
530 NEXT C: LET G=G(D): LET G(
D)=B: GOSUB 210: LET V=P: LET
G(D)=G: RETURN
```



```
410 V=H:FOR A=X(F)TO 0 STEP -1:
X=M(A,F):IF FNX(X)<0 THEN NEXT:
L=1:RETURN
420 B=F:F=X:GOSUB 210:V=P:F=B:L
=1:A=0:NEXT:RETURN
510 V=E:FOR C=1 TO 4:G=G(C):IF
-B(G)>V THEN 530
520 FOR A=0 TO Z(G):X=M(A,G):IF
 FNX(X) OR X=F THEN NEXT:GOTO 5
30
528 V=B(X)-B(G):D=C:B=X
530 NEXT:G=G(D):G(D)=B:GOSUB 21
0:V=P:G(D)=G:IF SG=1 THEN G(D)=
B:C=D
540 RETURN
```



```
410 V=H:FOR A=X(F) TO 0 STEP-1:
X=M(A,F)·IF FNX(X)<0 THEN NEXT·
L=1:RETURN
420 B=F:F=X·GOSUB 210:V=P·F=B·L
=1:A=0:NEXT·RETURN
```

```
520 FOR A=0 TO Z(G):X=M(A,G):IF
 FNX(X) OR X=F THEN NEXT:GOTO 5
30
528 V=B(X)-B(G):D=C:B=X
530 NEXT:G=G(D):G(D)=B:GOSUB 21
0:V=P:G(D)=G:IF SG=1 THEN G(D)=
B:C=D
540 RETURN
```



```
410 V = H: FOR A = X(F) TO 0 ST
EP  - 1:X = M(A,F): IF  FN X(X)
 < 0 THEN  NEXT :L = 1: RETURN
420 B = F:F = X: GOSUB 210:V =
P:F = B:L = 1:A = 0: NEXT : RET
URN


510 V = E: FOR C = 1 TO 4:G = G
(C): IF  - B(G) > V THEN 530
520  FOR A = 0 TO Z(G):X = M(A,
G): IF  FN X(X) OR X = F THEN
NEXT : GOTO 530
528 V = B(X) - B(G):D = C:B = X

530  NEXT :G = G(D):G(D) = B: G
OSUB 210:V = P:G(D) = G: IF SG
= 1 THEN G(D) = B:C = D:
540  RETURN
```



Essas sub-rotinas são usadas apenas quando o computador está nos menores níveis de dificuldade e analisam apenas o próximo movimento.

As linhas 410 e 420 analisam todos os movimentos possíveis para a raposa, usando o mapa da matriz **M**, montada a partir da linha 2110. A sub-rotina devolve um valor P (configuração após a melhor jogada) e um valor V (avaliação do melhor resultado).

As linhas 510 e 530 funcionam de maneira similar às anteriores, só que se encarregam dos gansos. As matrizes P e V são determinadas da forma já descrita. Nos dois casos, a instrução **GOSUB 210** escolhe o melhor movimento entre os que são avaliados.

A TABELA DE POSIÇÕES



```
1510 GOSUB 210: LET C=P
1520 LET C=C-INT ((C/HF+C)-C)*H
F: IF C<0 OR C>=HF THEN  GOTO 1
520
1550 FOR A=C+1 TO C+4: IF R(A)<
>0 AND R(A)<>P THEN  NEXT A: RE
TURN
1560 LET R(A)=P: LET S(A)=V: RE
TURN
1610 GOSUB 210: LET C=P
1620 LET C=C-INT ((C/HF+C)-C)*H
F: IF C<0 OR C>=HF THEN  GOTO 1
620
1650 FOR A=C+1 TO C+4: IF R(A)<
>0 AND R(A)<>P THEN  NEXT A: LE
T V=0: RETURN
1660 LET V=S(A)*(R(A)=P): RETUR
N
```



```
1510 GOSUB 210:C=P
1520 C=C-INT((C/HF+C)-C)*HF:IF
C<0 OR C>=HF THEN 1520
1550 FOR A=C TO C+C:IF R(A)<>0
AND R(A)<>P THEN NEXT:RETURN
1560 R(A)=P:S(A)=V:A=C+C:NEXT:R
ETURN
1610 GOSUB 210:C=P
1620 C=C-INT((C/HF+C)-C)*HF:IF
C<0 OR C>=HF THEN 1620
1650 FOR A=C TO C+C:IF R(A)<>0
AND R(A)<>P THEN NEXT:V=0:RETUR
N
1660 V=-S(A)*(R(A)=P):A=C+C:NEX
T:RETURN
1810 SCREEN 1,0:PUT(XX,YY)-(XX+
19,YY+19),SQ,NOT:FOR Z=1 TO 100
:NEXT
1820 PUT(XX,YY)-(XX+19,YY+19),S
Q,NOT
1830 K$=INKEY$:IF K$="^" AND YY
>8 AND XX>8 THEN YY=YY-20:XX=XX
-20:GOTO 1810
1840 IF K$=CHR$(10) AND YY<129
AND XX<129 THEN YY=YY+20:XX=XX+
20:GOTO 1810
1850 IF K$=CHR$(8) AND XX>28 TH
EN XX=XX-40:GOTO 1810
1860 IF K$=CHR$(9) AND XX<128 T
HEN XX=XX+40:GOTO 1810
1870 IF K$=CHR$(13) THEN RETURN
1875 IF K$="Q" THEN YY=0:XX=-12
:RETURN
1880 GOTO 1810
```



```
1510 GOSUB 210:C=P
1520 C=C-INT((C/HF+C)-C)*HF:IF
C<0 OR C>=HF THEN 1520
1550 FOR A=C TO C+C:IF R(A)<>0
AND R(A)<>P THEN NEXT:RETURN
1560 R(A)=P:S(A)=V:A=C+C:NEXT:R
ETURN
1610 GOSUB 210:C=P
1620 C=C-INT((C/HF+C)-C)*HF:IF
C<0 OR C>=HF THEN 1620
1650 FOR A=C TO C+C:IF R(A)<>0
```



```
AND R(A)<>P THEN NEXT·V=0 RETUR
N
1660 V=-S(A)*(R(A)=P)·A=C+C NEX
T·RETURN
1810 LINE (XX,YY-2)·(XX+19,YY+1
7),14,BF·FOR Z=1 TO 100·NEXT
1820 LINE (XX,YY-2)-(XX+19,YY+1
7),3,BF·FOR Z=1 TO 200·NEXT
1830 K$=INKEY$·IF K$=CHHS(30) A
ND YY>18 AND XX>18 THEN YY=YY-2
0:XX=XX-20 GOTO 1810
1840 IF K$=CHR$(31) AND YY<141
AND XX<141 THEN YY=YY+20 XX=XX+
20·GOTO 1810
1850 IF K$=CHR$(29) AND XX>38 T
HEN XX=XX-40 GOTO 1810
1860 IF K$=CHR$(28) AND XX<138
THEN XX=XX+40·GOTO 1810
1870 IF K$=CHR$(13) THEN RETURN
1880 GOTO 1810
```





```
1510  GOSUB 210 C = P
1520 C = C -  INT ((C / HF + C)
 - C) * HF· IF C < 0 OR C >   =
HF THEN 1520
1550  FOR A = C TO C + C: IF R(
A) <   > 0 AND R(A) <   > P THEN
 NEXT · RETURN
1560 R(A) = P:S(A) = V:A = C +
C: NEXT : RETURN
1610  GOSUB 210·C = P
1620 C = C -  INT ((C / HF + C)
 - C) * HF· IF C < 0 OR C >   =
HF THEN 1620
1650  FOR A = C TO C + C: IF R(
A) <   > 0 AND R(A) <   > P THEN
 NEXT ·V = 0· RETURN
1660 V = S(A) * (R(A) = P)·A =
C + C: NEXT : RETURN
1810 X0 = XX:Y0 = YY
1820  HCOLOR= 0· GOSUB 1890· HC
OLOR= 3·X0 = XX Y0 = YY· GOSUB
1890
1830  GET K$: IF K$ = "A" AND Y
Y > Y1 + 5 AND XX > X1 + 5 THEN
 YY = YY - 20:XX = XX - 20: GOT
O 1820
1840  IF K$ = "Z" AND YY < Y2 ·
 38 AND XX < X2 - 38 THEN YY =
YY + 20:XX = XX + 20: GOTO 1820

1850  IF K$ =  CHR$ (8) AND XX
> X1 + 21 THEN XX = XX - 40: GO
TO 1820
1860  IF K$ =  CHR$ (21) AND XX
 < X2 - 45 THEN XX = XX + 40: G
OTO 1820
1870  IF K$ =  CHR$ (13) THEN
HCOLOR= 0: GOSUB 1890: RETURN
1880  GOTO 1820
1890  HPLOT X0 + 1,Y0 + 1 TO X0
 + 19,Y0 + 1 TO X0 + 19,Y0 + 18
 TO X0 + 1,Y0 + 18 TO X0 + 1,Y0
 + 1: RETURN
```



Nos níveis de maior dificuldade será necessário usar o algoritmo alfa-beta — veja o primeiro artigo da série (página 872). Na verdade, ele já foi digitado como parte das rotinas de movimentação da raposa e dos gansos.

Antes de utilizá-lo, porém, o programa verifica se ele é necessário no momento — será o nível de dificuldade suficientemente alto para justificar o seu emprego?

A rotina das linhas 1110 a 1150 cuida da raposa, enquanto a rotina das linhas 1310 a 1350 dedica-se ao quarteto de gansos.

O algoritmo é aplicado quando se executa o último teste **IF**, no fim das linhas 1150 e 1350, depois de **V(M)** ter sido adequadamente definida, de acordo com o nível de dificuldade.

O algoritmo alfa-beta é mais eficiente quando o computador discrimina quais são os melhores movimentos realizados inicialmente — para os gansos, o quadrado de número mais alto de cada fileira de quatro; para a raposa, o quadrado aberto a ela mais próximo do lado dos gansos.

O algoritmo alfa-beta é usado em conjunto com uma tabela construída em função de movimentos que já foram considerados. Assim, o computador pode se decidir mais rapidamente, quando se trata de uma jogada já estudada. Quanto maior for o tamanho da tabela que é possível montar no computador, mais rápido será o processamento.

Inicializada nas linhas 2500, 2750 e 2800, a tabela tem seu tamanho ideal definido por meio de valores teóricos. Ela foi dimensionada, no entanto, no limite da memória RAM disponível em cada microcomputador.

A tabela é zerada nas linhas 1110 e 1310; o conteúdo é verificado nas linhas 1122 e 1322 e definido nas linhas 1172 e 1372

O TRS-Color e o MSX usam um cursor para indicar as jogadas a serem executadas com as setas. O cursor do Apple é movido para a esquerda e para a direita com as setas, e para cima e para baixo com as letras A e Z.

A ARANHA MARCIANA (1)

- O "ENREDO"
- DEFINIÇÃO DOS GRÁFICOS
- FREDDY, AS FLECHAS, OS BALÕES E A ARANHA
- INICIALIZAÇÃO DO JOGO

Uma enorme e assustadora aranha está pronta a atacar Freddy. Sem uma boa dose de precisão e sangue-frio, ele não escapará. Mas não se assuste: tudo não passa de um terrível pesadelo.

O jogo *A Aranha Marciana* será montado em dois artigos. Neste, os desenhos são definidos e o programa inicializado. No próximo, o jogo passa a funcionar, com a adição das últimas rotinas.

O JOGO

O ponto inicial para a montagem de um jogo é a criação de algum tipo de enredo, ou mesmo de um personagem em torno do qual a ação se desenvolva.

Neste jogo, nosso personagem é Freddy, um limpador de janelas que tem um medo doentio de aranhas. Ele procurou vários especialistas que tentaram, em vão, curá-lo da fobia. O problema chegou a uma tal gravidade que Freddy passou a ter sempre o mesmo pesadelo. Nele aparecem uma aranha marciana — particularmente grande, faminta e de aparência horrível —, uma coleção de balões, um arco e uma flecha.

Com frequência, ele acorda banhado em suor, após ter sonhado que estava em sua escada, tentando desesperadamente flechar balões que, se atingissem a gaiola da aranha, acima de sua cabeça, destravariam a porta que aprisiona o monstro. Ajude Freddy a estourar os balões, ou ele terá um trágico fim, como almoço de uma horrenda aranha!

Temos, assim, um enredo. Com relação aos pontos, eles serão dados para cada balão estourado. Mas, mesmo que Freddy consiga interceptar todos os balões, seu problema não estará resolvido a tortura continuará num nível de dificuldade ainda maior com balões mais rápidos. Se deixar que três deles passem, a aranha marciana sairá da gaiola.

O PROGRAMA

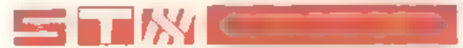
Nosso jogo requer a animação de quatro figuras: a aranha, que se movimenta na vertical e na horizontal; Freddy, que se move só na vertical; o balão, que pode aparecer em qualquer



ponto da parte inferior da tela, mas que também se move na vertical, e a flecha. Esta normalmente se move na horizontal, mas acompanha o movimento vertical de Freddy.

As figuras de maior interesse são a aranha e o balão. Muitas variáveis estarão associadas a elas, e será útil armazená-las em uma matriz unidimensional e usar uma constante como referência.

Além do movimento, precisamos definir o desenho de todas as diferentes figuras — animadas ou não — que aparecerão na tela. Como nos melhores jogos de computador, elas devem ser coloridas. Freddy será montado em um bloco gráfico de três por dois caracteres. Usaremos outros blocos de dois por dois caracteres para definir a aranha, um balão inteiro e um balão estourado, dois caracteres para o desenho da flecha e mais dois para o da escada. Assim, teremos no final cerca de 26 caracteres.



A primeira parte do programa define os gráficos e inicializa o jogo.

DEFINIÇÃO DOS GRÁFICOS



```
1000 DIM b(6): DIM s(7)
1010 LET xpos=1: LET ypos=2: LE
T colour=3: LET points=4: LET c
```

```
ount=5· LET maxcount=6
1020 LET xinc=3: LET yinc=4: LE
T picture=7
1030 LET dest=65288
1040 FOR i=0 TO 26*8-1: READ j:
 POKE dest+i,j: NEXT i
1050 DATA 15,63,127,255,255,255
,255,127
1060 DATA 240,252,254,255,255,2
55,255,254
1070 DATA 127,63,63,31,15,7,3,6
1080 DATA 254,252,252,248,240,2
24,192,96
1090 DATA 32,96,255,255,96,32,0
,0
1100 DATA 5,10,252,252,10,5,0,0
1110 DATA 1,64,17,40,16,0,0,161
1120 DATA 128,2,136,20,8,0,0,13
3
1130 DATA 161,0,0,16,50,17,64,1
1140 DATA 133,0,0,8,20,136,2,12
8
1150 DATA 48,48,48,48,111,111,4
8,48
1160 DATA 12,12,12,12,246,246,1
2,12
1170 DATA 7,31,49,57,127,112,23
7,255
1180 DATA 224,248,140,204,254,1
4,183,255
1190 DATA 127,59,51,99,115,35,6
,12
1200 DATA 254,108,102,51,49,25,
24,48
1210 DATA 7,31,49,51,127,112,23
5,255
1220 DATA 224,248,140,156,254,1
4,215,255
1230 DATA 127,51,50,27,25,50,11
2,224
1240 DATA 254,204,108,102,54,22
,3,6
1250 DATA 15,31,19,55,55,63,63,
15
1260 DATA 240,248,248,252,252,2
52,252,240
1270 DATA 251,219,139,219,219,2
51,247,239
1280 DATA 252,254,254,254,254,2
54,254,252
1290 DATA 95,127,31,31,31,63,12
7,127
1300 DATA 188,188,188,188,188,1
24,252,248
1310 LET hiscore=0
1320 RETURN
```



```
1000 DIM B(6),S(7),NU$(9)
1010 XP=1:YP=2:PO=4:CT=5:MC=6
1020 XI=3:YI=4:PI=7
1022 DIM AD(14),EF(4),E(4),GJ(7
),KL(4),MP(7),QT(7),UZ(14),SP(7
),S1(4),S2(7),BL(4)
1023 PMODE 4,1:PCLS1:SG=PEEK(18
8)*256
1024 GOSUB 1500
1025 GET(0,0)-(15,23),AD,G:PCLS
1
1026 GOSUB 1520
1027 GET(0,0)-(15,7),EF,G:GET(0
,0)-(7,7),E,G·PCLS1
1028 GOSUB 1500
1029 GET(0,0)-(15,15),GJ,G:PCLS
1
1030 GOSUB 1520
1031 GET(0,0)-(15,7),KL,G:PCLS1
1032 GOSUB 1500
1033 GET(0,0)-(15,15),MP,G:PCLS
1
1034 GOSUB 1500
1035 GET(0,0)-(15,15),QT,G:PCLS
1
1036 GOSUB 1500:SG=SG+512:GOSUB
 1520:SG=SG-512
1037 GET(0,0)-(15,23),UZ,G:PCLS
1
1038 GET(0,0)-(15,15),SP,G
1039 GET(0,0)-(7,7),S1,G
1040 GET(0,0)-(15,7),S2,G
1041 PCLS0:GET(0,0)-(7,7),BL,G
1050 DATA 15,63,127,255,255,255
,255,127
1060 DATA 240,252,254,255,255,2
55,255,254
1070 DATA 127,63,63,31,15,7,3,6
1080 DATA 254,252,252,248,240,2
24,192,96
1090 DATA 32,96,255,255,96,32,0
,0
1100 DATA 5,10,252,252,10,5,0,0
1110 DATA 1,64,17,40,16,0,0,161
1120 DATA 128,2,136,20,8,0,0,13
3
1130 DATA 161,0,0,16,40,17,64,1
1140 DATA 133,0,0,8,20,136,2,12
8
1150 DATA 48,48,48,48,111,111,4
8,48
1160 DATA 12,12,12,12,246,246,1
2,12
1170 DATA 7,31,49,57,127,112,23
7,255
1180 DATA 224,248,140,204,254,1
4,183,255
1190 DATA 127,59,51,99,115,35,6
,12
1200 DATA 254,108,102,51,49,25,
24,48
1210 DATA 7,31,49,51,127,112,23
5,255
1220 DATA 224,248,140,156,254,1
4,215,255
1230 DATA 127,51,50,27,25,50,11
2,224
1240 DATA 254,204,108,102,54,22
,3,6
1250 DATA 15,31,19,55,55,63,63,
15
1260 DATA 240,248,248,252,252,2
52,252,240
1270 DATA 251,219,139,219,219,2
51,247,239
1280 DATA 252,254,254,254,254,2
54,254,252
1290 DATA 95,127,31,31,31,63,12
7,127
1300 DATA 188,188,188,188,188,1
24,252,248
1310 HS=0
1320 RETURN
1500 FOR CL=0 TO 1:FOR CH=0 TO
1:FOR L=0 TO 7:READ J:POKE SG+C
L*256+CH+L*32,255-J:NEXT L,CH,C
L
1510 RETURN
1520 FOR CH=0 TO 1:FOR L=0 TO 7
:READ J:POKE SG+CH+L*32,255-J:N
EXT L,CH
1530 RETURN
1600 DATA R6D8L6U8BR8,BR6ND8BR2
,R6D4L6D4R6BR2BU8,R6D4NL3D4NL6B
R2BU8,D4R6D4U8BR2,NR6D4R6D4L6BE
8
1610 DATA D8R6U4L6U4BR8,R6ND8BR
2,R6D8L6U8D4R6U4BR2,D4R6D4U8L6B
R8
1620 FOR I=0 TO 9:READ NU$(I):N
EXT
1625 DRAW"C1;S2"
1630 RETURN
1650 N$=STR$(NU):FOR I2=2 TO LE
N(N$)
1660 DI=ASC(MID$(N$,I2,1))-48:D
RAW NU$(DI)+"BR2":NEXT I2:RETUR
N
1700 COLOR 0:LINE (178,2)-(200,
7),PSET,BF:NU=HS:DRAW"C1:BM178,
2":GOSUB 1650:RETURN
```



```
1000 DIM B(6),S(7):SCREEN2,2
1005 A1=1·A2=2:BA=3:BE=4·FD=5·F
L=6
1010 XP=1:YP=2:PO=4:CT=5·MC=6
1020 XI=3:YI=4·PI=7
1030 FOR I=1 TO 6:A$=""
1040 FOR J=1 TO 32
1050 READ A:A$=A$+CHR$(A)
1060 SPRITE$(I)=A$
1070 NEXT:NEXT
1100 DATA 0,0,0,0,7,13,31,18,15
,10,10,18,10,2,4,0,0,0,0,0,208,
176,248,72,240,80,80,72,80,64,1
28,0
1110 DATA  0,0,0,0,7,13,31,18,1
5,10,10,9,8,8,4,0,0,0,0,0,208,1
76,248,72,240,80,80,144,16,16,8
,0
1120 DATA  0,0,0,3,15,31,31,31,
15,15,7,3,3,1,0,0,0,0,0,192,240
,248,248,248,240,240,224,192,19
2,128,0,0
1130 DATA  0,0,0,0,8,4,0,13,1,0
,4,4,8,0,0,0,0,0,0,16,32,0,0,12
8,152,0,64,32,0,0,0,0
1140 DATA  0,3,7,7,3,1,7,15,15,
15,15,7,3,2,2,1,0,192,224,224,1
92,128,224,240,240,240,240,224,
192,192,192,128
1150 DATA  0,0,0,0,0,0,16,112,2
55,112,16,0,0,0,0,0,0,0,0,0,0,0
,8,16,240,16,8,0,0,0,0,0
1310 HS=0
1320 RETURN
```

Esta parte do programa lê os dados das linhas **DATA** para montar os blocos gráficos (UDG ou sprites) do balão e da aranha nas matrizes **B** (ou **b**) e **S** (ou **s**), que são dimensionadas na linha 1000.

As linhas 1010 e 1020 definem os valores iniciais para os apontadores das



matrizes, antes dos UDG ou sprites serem montados. As linhas 1030 a 1040 (até 1070, no MSX) fazem a leitura dos dados em **DATA** e montam os blocos gráficos. Finalmente, a linha 1310 define o recorde como 0. Essa linha é executada apenas uma vez. O programa para o TRS-Color inclui uma rotina a mais, que começa na linha 1600. Sua função é desenhar números na tela de alta resolução.

INICIALIZAÇÃO DO JOGO

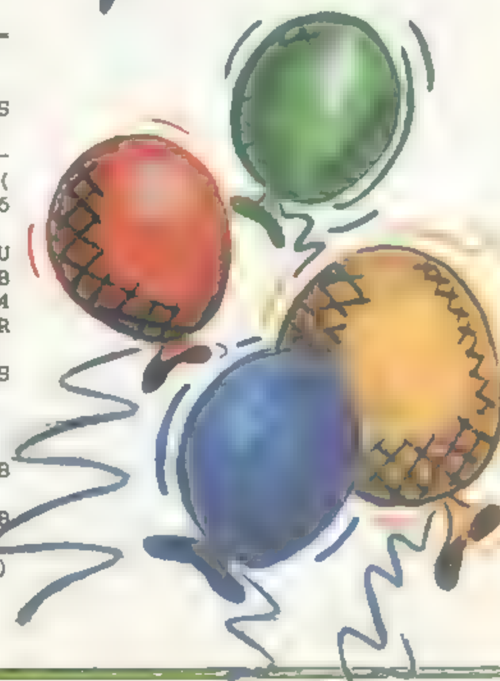
S

```
3000 LET score=0· LET level=1
3010 LET my=15
3020 LET bl=15+5*level· LET ax=
29: LET ay=16: LET dead=0: LET
props=3
3090 GOSUB 5000
3150 PAPER 0· BORDER 0: CLS
3160 FOR x=0 TO 28: PRINT AT 3,
x; INK 0; PAPER 6;" ";AT 0,x;"
": NEXT x: GOSUB 6000
3170 POKE 23607,60: PRINT AT 0,
0; INK 0; PAPER 6;"N=";level;"
 B=";bl;"  ";"SC=";score;AT 0,2
0;"RE=";hiscore
3180 POKE 23607,252
3190 FOR y=5 TO 21: PRINT AT y,
30; INK 6;"kl"· NEXT y
3200 POKE 23607,252
3210 GOSUB 4000
3220 GOSUB 4200
3240 RETURN
```



T

```
3000 SC=0:LV=1
3010 MY=15
3020 BL=15+5*LV:AX=29:AY=16:DD=
0:PP=3
3090 GOSUB 5000
3150 PMODE 4,1:COLOR 0.1:PCLS:S
CREEN 1.1
3160 FOR X=0 TO 28:PUT(X*8,24)-
(X*8+7,31),BL,PSET:PUT(X*8,0)-(
X*8+7,7),BL,PSET:NEXT X:GOSUB 6
000
3165 DRAW"S4;BM2,2;C1;D4R3BR2BU
1R2BU1L2;BM48,2;D4R4U2L4R3U2L3B
R6BD2R2BD1L2;BM100,2;L4D2R4D2L4
BR6NR4U4R4;BM160,2;D4U2R4D2U4BR
2R4L2D4L2R4;S2"
3170 DRAW"C1;BM14,2;":NU=LV:GOS
UB 1650
3171 DRAW"BM58,2;":NU=BL·GOSUB
1650
3172 DRAW"BM114,2;":NU=SC:GOSUB
 1650
3173 DRAW"BM178,2;":NU=HS:GOSUB
 1650
3190 FOR Y=5 TO 21:PUT(240,Y*8)
-(255,Y*8+7),KL,PSET:NEXT Y
3210 GOSUB 4000
```



```
3220 GOSUB 4200
3240 RETURN
```

MSX

```
3000 SC=0·LV=1
3010 MY=16
3020 BL=15+5*LV:AX=29:AY=16·DD=
0 PP=3
3090 GOSUB 5000
3160 LINE (0,0)-(231,31),6,BF:L
INE (0,7)-(80,24),14,BF:LINE (8
7,7)-(152,24),14,BF:LINE (159,7
)-(224,24),14,BF
3170 GOSUB 1700
3180 GOSUB 6000
3190 LINE (240,31)-(240,190),1:
LINE (255,31)-(255,190),1·FOR I
=2 TO 15:LINE (240,I*16)-(255,I
*16),1·NEXT
3210 GOSUB 4000
3220 GOSUB 4200
3230 RETURN
```

As linhas 3000, 3010 e 3020 zeram o placar, inicializam o nível de dificuldade em 1 e definem uma série de variáveis auxiliares. A linha 3150 determina as cores da tela (menos no MSX) e as linhas 3160 a 3170 exibem o placar e outras informações necessárias.

O programa do Spectrum inclui um **POKE** na linha 3170 e outro na 3180. A posição de memória 23607 guarda o apontador do conjunto de caracteres. Normalmente, ela tem o valor 60, que aponta para o conjunto de caracteres da ROM. Nesse programa, entretanto, o valor colocado no endereço 23607 para indicar a posição dos caracteres é 252. Isso permite ao computador usar as letras minúsculas, além das maiúsculas e dos números, como gráficos.

O programa reserva uma área da memória usando **CLEAR** e ali coloca os UDG. Devido ao seu efeito sobre as sub-rotinas, esse comando deve ser utilizado no programa principal, o que faremos na segunda parte do jogo. Portanto, se você quiser executar as linhas aqui apresentadas, digite antes **CLEAR** 65287.

A linha 3190 se incumbe de desenhar a escada, e as sub-rotinas chamadas nas linhas 4000 e 4200 desenham, respectivamente, Freddy e a aranha.



FREDDY E A FLECHA

S

```
4000 INK 7· PRINT AT my,30;"uv"
;AT my+1,30;"wx";AT my+2,30;"yz
": IF ax=29 THEN  PRINT AT ay,a
x;"e";
4010 RETURN
4110 INK 7· PRINT AT ay,ax;"ef"
: RETURN
```



As figuras montadas pelo programa tal como aparecem na tela dos microcomputadores da linha Spectrum.



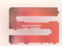
```
4000 PUT(240,MY*8)-(255,MY*8+23
),UZ,PSET:IF AX=29 THEN PUT(AX*
8,AY*8)-(AX*8+7,AY*8+7),E,PSET
4010 RETURN
4110 PUT(AX*8,AY*8)-(AX*8+15,AY
*8+7),EF,PSET:RETURN
```



```
4000 PUT SPRITE 2,(240,MY*8),1,
FD·IF AX=29 THEN PUT SPRITE 3,(
AX*8,AY*8),4,FL
4010 RETURN
4110 PUT SPRITE 3,(AX*8,AY*8).4
,FL·RETURN
```

Essa rotina desenha Freddy sobre a escada e sua flecha. A posição do nosso personagem é determinada pelo valor de **MY**. A flecha, por sua vez, é desenhada pela linha 4110 na posição da tela indicada pelos valores de **AX** e **AY**.

A ARANHA MARCIANA



```
4200 IF s(picture)=1 THEN  GOTO
 4250
4210 PRINT AT s(ypos),s(xpos);"
mn",AT s(ypos)+1,s(xpos);"op":
RETURN
4250 PRINT AT s(ypos),s(xpos);"
qr",AT s(ypos)+1,s(xpos);"st":
RETURN
```



```
4200 X2=S(XP)*8:Y2=S(YP)*8:IF S
(PI)=1 THEN 4250
4210 PUT(X2,Y2)-(X2+15,Y2+15),M
P,PSET:RETURN
4250 PUT(X2,Y2)-(X2+15,Y2+15),Q
T,PSET:RETURN
```



```
4200 X2=S(XP)*8·Y2=S(YP)*8:IF S
(PI)=1 THEN 4250
4210 PUT SPRITE 1,(X2,Y2),12,A1
:RETURN
4250 PUT SPRITE 1,(X2,Y2),12,A2
:RETURN
```

A rotina que desenha a aranha se assemelha à anterior, exceto num detalhe: duas imagens são usadas para essa figura. Colocadas alternadamente no mesmo lugar da tela, pelas linhas 4210 e 4250, simulam o movimento das pernas da aranha.

SOBEM OS BALÕES



```
4300 PRINT AT b(ypos),b(xpos);
BRIGHT 1; INK b(colour);"ab";AT
 b(ypos)+1,b(xpos);"cd": RETURN

5000 LET range=INT (RND*6)
5010 LET b(xpos)=(4*range)+INT
(RND*4)
5020 LET b(ypos)=20
5030 LET b(maxcount)=5-level
5040 LET b(count)=1
5050 LET b(colour)=INT (RND*5)+
3
5060 LET b(points)=10-range
5070 RETURN
```



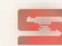
```
4300 X2=B(XP)*8:Y2=B(YP)*8:PUT(
X2,Y2)-(X2+15,Y2+23),AD,PSET:RE
TURN
5000 RG=RND(6)-1
5010 B(XP)=(4*RG)+RND(4)-1
5020 B(YP)=20
5030 B(MC)=5-LV
5040 B(CT)=1
5060 B(PO)=10-RG
5070 RETURN
```



```
4300 X2=B(XP)*8:Y2=B(YP)*8·PUT
SPRITE 4,(X2,Y2),9,BA·RETURN
5000 RG=INT(RND(1)*6)
5010 B(XP)=(4*RG)+INT(RND(1)*5)
5020 B(YP)=20
5030 B(MC)=5-LV
5040 B(CT)=1
5050 B(PO)=10-RG
5070 RETURN
```

A linha 4300 simplesmente desenha os balões na tela. As linhas restantes encarregam-se de inflar um balão toda vez que um outro estourar ou alcançar a gaiola da aranha (os balões podem aparecer em qualquer um dos seis pontos da parte inferior da tela, flutuando em seguida para cima, na direção vertical. A variável **MAXCOUNT** (ou, ainda, **maxcount** ou **MC**, conforme o computador) determina o quanto o balão se moveu, ou seja, o nível que ele atingiu. Em seguida, o programa define a cor do balão e seu valor em pontos. Este dependerá de sua proximidade da escada de Freddy.

AS PORTAS



```
6000 IF level<>1 THEN  POKE 236
07,60: PRINT AT s(ypos),s(xpos)
;"  ";AT s(ypos)+1,s(xpos);"  "
: POKE 23607,252
6010 FOR x=10 TO 30 STEP 9
6020 PRINT  INK 6;AT 1,x;"";AT
2,x;""
6030 NEXT x
6040 LET s(xpos)=1: LET s(ypos)
=1: LET s(xinc)=1: LET s(yinc)=
0: LET s(count)=4: LET s(maxcou
nt)=4: LET s(picture)=1
6050 RETURN
```



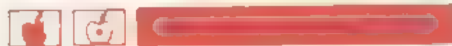
```
6000 IF LV<>1 THEN X2=S(XP)*8:Y
2=S(YP)*8:PUT(X2,Y2)-(X2+15,Y2+
15),SP,PSET
6010 FOR X=10 TO 30 STEP 9
6020 PUT(X*8,8)-(X*8+7,15),BL,P
SET:PUT(X*8,16)-(X*8+7,23),BL,P
SET
6030 NEXT X
6040 S(XP)=1:S(YP)=1:S(XI)=1:S(
```

```
YI)=0·S(CT)=4:S(MC)=4:S(PI)=1
6050 RETURN
```



```
6000 S(XP)=1·S(YP)=1·S(XI)=1 S(
YI)=0·S(CT)=4·S(MC)=4·S(PI)=1
6010 RETURN
```

Três portas fecham a gaiola da aranha, para mantê-la presa. Os usuários do Spectrum notarão que o caractere listado como ? e, de fato, um quadradinho gráfico, obtido com a tecla 8 em modo gráfico.



A primeira seção do programa define os gráficos e inicializa o jogo.

A INICIALIZAÇÃO

```
1000 DIM B(6),S(7),NUS(9): SCA
LE= 1· ROT= 0· HCOLOR= 3
1010 XP = 1:YP = 2:PO = 4:CT =
5·MC = 6
1020 XI = 3:YI = 4:PI = 7
1025 A1 = 1:A2 = 2:BA = 3:BE =
4 FD = 5·FL = 6
1030 DATA 6,0,14,0,75,0,136,
0,1,1,21,1,158,1
1040 DATA 45,45,45,45,77,58,
63,63,63,63,63,63,55,173,240
1050 DATA 43,45,45,44,45,53,4
5,37,45,62,54,63,63,63,63,63,63
,188
1060 DATA 10,62,62,62,54,109,
145,37,37,36,36,100,73,17,54,54

1070 DATA 54,53,109,193,193,1
93,45,36,39,39,36,0
1080 DATA 45,45,45,45,117,57
,63,63,63,63,63,63,55,45,222
1090 DATA 45,45,37,45,45,46,4
5,44,53,55,62,63,63,63,63,63,39
,151
1100 DATA 49,54,62,62,62,79,7
3,72,1,193,193,39,39,108,73
1110 DATA 9,54,55,183,74,73,7
3,39,39,39,36,36,0
1120 DATA 45,45,45,53,63,63,
63,63,55,45,45,45,45,45,44,54
1130 DATA 63,63,63,63,63,63,5
5,45,45,45,45,45,45,45,44,54,63

1140 DATA 63,63,63,63,63,63,5
5,45,45,45,45,45,45,45,53,63
1150 DATA 63,63,63,63,63,63,5
5,45,45,45,45,45,45,45,53,63,63
,63
1160 DATA 63,63,63,63,46,45,
45,45,45,45,45,62,63,63,63,63,6
,3
1170 DATA 55,45,45,45,45,45,5
3,63,63,63,63,63,46,45,45,45,45
,62
1180 DATA 63,63,63,55,45,45,4
5,53,63,63,55,62,87,73,73,8,39,
4,0
1190 DATA 46,110,9,44,172,146
,57,223,219,59,191,146,45,37,77
,9,53,46,45,0
1200 DATA 45,45,45,53,63,63,
63,63,46,45,45,45,45,44,54,63
1210 DATA 63,63,63,63,46,45,4
5,45,45,62,63,63,63,55,45,45
1220 DATA 45,45,222,63,63,62,
63,63,46,45,45,37,45,45,45,45,5
3,63,63
1230 DATA 63,63,62,63,63,63,4
6,45,45,45,37,45,45,45,53,63
1240 DATA 63,63,62,63,63,63
,63,46,45,45,45,45,45,44,45,53
1250 DATA 63,55,62,63,63,63,6
3,60,54,45,45,45,45,45,62,63,63

1255 DATA 63,63,46,45,45,45,5
3,63,63,63,63,46,45,45,45,62,63
,63,55
1260 DATA 45,45,45,62,63,63,4
6,45,45,62,63,63,55,45,45,45,5,
0
1270 DATA 45,44,54,119,33,36
,44,54,54,37,36,172,42,45,45,45
,45,5,0
1280 FOR I = 16384 TO 16816 R
EAD C: POKE I,C: NEXT
1290 P = 233· POKE P,64· POKE P
- 1,0
1300 HS = 0
1310 RETURN
```

Essa parte do programa lê os dados das linhas **DATA** para montar, nas matrizes **B** e **S** (dimensionadas na linha 1000), os blocos gráficos do balão e da aranha.

As linhas 1010 a 1025 determinam os valores iniciais para os apontadores das matrizes. As linhas 1280 a 1290 fazem a leitura dos dados em **DATA** (linhas 1030 a 1270) e a montagem dos blocos gráficos. Finalmente, a linha 1310 define o recorde como 0. Essa linha será executada apenas uma vez.

Adicione agora este programa:

```
3000 SC = 0:LV = 1
3010 MY = 15
3020 BL = 15 + 5 * LV:AX = 28:A
Y = 16:DD = 0:PP = 3
3090 GOSUB 5000
3150 HGR : GOSUB 1700
3160 FOR Y = 0 TO 3: HPLOT 0,Y
 TO 234,Y: HPLOT 0,Y + 23 TO 23
4,Y + 23: NEXT
3170 FOR X = 0 TO 3: HPLOT X +
 77,3 TO X + 77,23: HPLOT X + 1
54,3 TO X + 154,23: HPLOT X + 2
31,3 TO X + 231,23: NEXT
3175 GOSUB 6000
3180 HPLOT 235,25 TO 235,160:
HPLOT 255,25 TO 255,160
3210 GOSUB 4000
3220 GOSUB 4200
3240 RETURN
```

As linhas 3000, 3010 e 3020 zeram o placar, inicializam o nível de dificuldade em 1 e definem uma série de variáveis auxiliares necessárias ao jogo. As linhas 3160 a 3180 mostram o placar e outras informações. A linha 3190 desenha a escada, e as sub-rotinas chamadas nas linhas 4000 e 4200 desenham, respectivamente, Freddy e a aranha.

A rotina seguinte desenha Freddy na escada e sua flecha. A posição do nosso personagem é determinada pelo valor de **MY**. A flecha, por sua vez, é desenhada pela linha 4110 na posição da tela indicada pelos valores de **AX** e **AY**.

```
4000 DRAW FD AT 240,MY * 8: IF
 AX = 28 THEN DRAW FL AT AX *
8,AY * 8
4010 RETURN
4110 DRAW FL AT AX * 8,AY * 8·
 RETURN
```

A ARANHA MARCIANA

```
4200 X2 = S(XP) * 8·Y2 = S(YP)
* 8· IF S(PI) = 1 THEN 4250
4210 DRAW A1 AT X2,Y2 RETURN

4250 DRAW A2 AT X2,Y2: RETURN
```

A rotina que desenha a aranha é muito parecida com a anterior, exceto por um detalhe: duas imagens são usadas para essa figura (**A1** e **A2**). Elas são colocadas alternadamente no mesmo lugar da tela, pelas linhas 4210 e 4250, simulando que as pernas da aranha se movem.

```
4300 X2 = B(XP) * 8·Y2 = (B(YP)
 + 1) * 8: HCOLOR= 0: DRAW BA A
T X2,Y2: HCOLOR= 3·Y2 = B(YP) *
 8· DRAW BA AT X2,Y2· RETURN
5000 RG = INT ( RND (1) * 6)
5010 B(XP) = (4 * RG) + INT (
RND (1) * 5)
5020 B(YP) = 20
5030 B(MC) = 5 - LV
5040 B(CT) = 1
5050 B(PO) = 10 - RG
5070 RETURN
```

A linha 4300 desenha os balões na tela. As linhas restantes encarregam-se de inflar um balão toda vez que um deles estourar ou alcançar a gaiola da aranha.

Os balões aparecem em qualquer um dos seis pontos da parte inferior da tela, flutuando em seguida verticalmente. A variável **MC** determina o nível que o balão atingiu. Por fim, o programa estabelece o valor, em pontos, do balão, segundo sua proximidade da escada.

A rotina final dessa parte do jogo é responsável pela contagem do número de portas (máximo de três) que estão mantendo a aranha dentro da gaiola.

```
6000 IF LV <  > 1 THEN X2 = S(
XP) * 8:Y2 = S(YP) * 8: DRAW AL
GUMACOISA
6040 S(XP) = 1:S(YP) = 1:S(XI)
= 1:S(YI) = 0:S(CT) = 4:S(MC) =
 4:S(PI) = 1
6050 RETURN
```

O JOGO DA VIDA

- AS REGRAS DA VIDA
- A DISPOSIÇÃO INICIAL DA COLÔNIA
- COMO CRIAR UMA COLÔNIA
- O PROGRAMA EM CÓDIGO

Programe seu micro para simular a eterna luta dos seres unicelulares pela vida. Depois, desafie um parceiro para jogar. O arranjo inicial da colônia de bactérias definirá o vencedor.

Neste fascinante jogo, o vídeo apresenta um mundo bidimensional, onde células podem viver, multiplicar-se e morrer. O *Jogo da Vida*, como é denominado, foi inventado por um cientista inglês há alguns anos, para ser jogado em um tabuleiro como o de xadrez. Em sua versão computadorizada, bastante popular entre os usuários de microcomputadores, a tela é dividida em um padrão quadriculado (invisível para o jogador). Cada quadradinho pode abrigar uma célula — uma bactéria, por exemplo. Essa célula terá, de acordo com sua disposição no diagrama, até um máximo de oito vizinhos.

As regras do *Jogo da Vida* determinam quando uma célula deve sobreviver ou morrer, e também, quando uma nova célula deve nascer. São elas:

- Uma célula nasce, sempre que houver um espaço cercado por exatamente três vizinhos.
- Uma célula consegue sobreviver na geração seguinte quando tem dois ou três vizinhos.
- As células que não se enquadrarem nas situações anteriores morrem. Em um espaço com mais de três vizinhos, por exemplo, supõe-se que faltará alimento ou oxigênio para todas as células.

Baseado nessas regras, o programa determina o futuro de cada quadrado na tela e mostra como a colônia inicial, montada pelo jogador, se desenvolve de geração para geração. Cada geração corresponde a um ciclo completo de cálculos para todo o quadriculado.

Dependendo do tamanho da grade, ou seja, do número de quadradinhos, o cálculo de uma geração será muito demorado se se utilizar um programa escrito em linguagem BASIC. Por isso, montamos o programa em linguagem de máquina, o que lhe permitirá observar uma geração da colônia a cada segundo, aproximadamente.

Como as gerações são mostradas em cores diferentes, o efeito visual é muito interessante. Você verá, na tela, como as colônias se espalham, mudam, fragmentam-se em unidades menores, morrem ou rejuvenescem.

A forma inicial da colônia é fundamental para o desenrolar do jogo. Alguns padrões condenam a colônia ao fim após algumas gerações; outros possibilitam sua sobrevivência por centenas e centenas de gerações. Certas composições determinam, ainda, que a colônia oscile entre um padrão e outro enquanto durar a simulação.

Cabe ao jogador estabelecer o padrão inicial, entrando as posições das primeiras células. Um dos objetivos do jogo consiste em criar uma colônia que dure o maior número possível de gerações (o programa informa quantas gerações se passaram). Entretanto, às vezes, vale a pena competir simplesmente para ver quem compõe a colônia com efeito mais interessante. Existem alguns padrões, por exemplo, que se deslocam em uma direção, recriando sua forma original a cada quatro ou cinco gerações, com um efeito adicional: a eliminação de todas as células que vão sendo encontradas no caminho.

Você poderá aproveitar esse efeito para desenvolver um jogo em que dois oponentes montam formas "devoradoras", ganhando aquele que destruir mais depressa a colônia do outro.

O PROGRAMA

Não entraremos em detalhe quanto ao funcionamento do programa em linguagem de máquina. Este é carregado e acionado por meio de um programa em linguagem BASIC que utiliza os comandos **POKE** e **USR**. Os códigos decimais correspondentes ao programa em são armazenados nas instruções **DATA** que começam na linha 500 (no Spectrum) ou 1000 (no TRS-Color). A parte do programa escrito em BASIC simplesmente lê esses códigos, colocando-os numa parte protegida da memória. Além disso, ela é responsável pela criação da tela e pela entrada da colônia inicial.

Este jogo é um interessante exemplo do uso de programas em código de máquina a partir de um programa em BASIC. Apresentamos aqui as versões para os micros da linha Spectrum e TRS-Color. Os programas para outras máquinas serão dados posteriormente.

```
[illegible] CLEAR 2867[illegible]: FOR N=USR [illegible]
  TO USR "A"+7[illegible]: READ A:
POKE N,A: NEXT N
6 DATA [illegible],24,60,[illegible]
[illegible]
[illegible] GOSUB 700
[illegible] POKE 2365[illegible]
```

```
BORDER [illegible]: INK [illegible]: PAPER [illegible]: CLS
[illegible] FOR N=[illegible] TO [illegible]: PRINT AT N,
0; PAPER 1;"        ": NEXT N
30 PLOT 63,0: DRAW 0,175:
DRAW 192,0: DRAW 0,-175: DRAW
-255,0: DRAW 0,175: DRAW 63,[illegible]
40 PRINT AT 2,1; PAPER 2; INK
7;" VIDA "; PAPER 0; INK 6;AT
5,1;" GER   ";AT 6,1;" 0000 "
70 RAND USR 28672: LET X=2[illegible]
LET Y=10
80 PRINT AT Y,X; OVER 1;"[illegible]":
FOR N=1 TO 10: NEXT N: PRINT
AT Y,X; OVER 1;" "
90 LET A$=INKEY$: IF A$="[illegible]"
THEN  GOTO 80
92 IF A$="0" THEN  GOTO 110
[illegible] IF CODE A$=13 THEN  PRINT
AT Y,X;CHR$ 144: POKE 30000+(
(Y-1)*23)+(X-8),144: GOTO 80
95 IF A$="M" THEN  PRINT AT Y
,X;" ": POKE 30000+((Y-1)*23)
+(X-8),32: GOTO 80
100 LET X=X-(A$="5")*(X>8)+(A$
="8")*(X<30): LET Y=Y-(A$="7")
*(Y>1)+(A$="6")*(Y<20): GOTO
80
110 OVER 0: RAND USR 28711
120 PRINT AT 21,1; FLASH 1;"QU
ALQUER TECLA PARA RECOMECAR":
FOR N=1 TO 200: NEXT N
130 LET A$=INKEY$: IF A$="[illegible]"
THEN  GOTO 130
140 GOTO 10
200 LET L=500: RESTORE [illegible]: FOR
N=28672 TO 28951 STEP 8
210 LET T=0: FOR D=0 TO [illegible]
220 READ A: POKE N+D,A: LET T=
T+A: NEXT D: READ A: IF A<>T
THEN  PRINT  FLASH 1;"ERRO DE
DADOS NA LINHA [illegible]
230 LET L=L+10
240 NEXT N
250 RETURN
500 DATA [illegible]
32,484
510 DATA 119,35,13,32,249,5,
242,6,701
520 DATA [illegible]
,34,422
530 DATA 2,113,33,48,117,34,
252,112,711
540 DATA 33,248,118,34,254,112
,201,243,1243
550 DATA 42,252,112,6,1,197,66
,6,704
560 DATA 8,80,62,22,215,123,
215,122,847
570 DATA 215,126,35,254,32,40,
10,214,926
580 DATA 142,[illegible]
,215,62,1198
590 DATA 144,215,4,120,254,31,
32,233,1033
600 DATA 193,4,120,254,21,32,
214,42,880
610 DATA 252,112,237,91,254,
112,229,213,1500
620 DATA [illegible]
113,770
630 DATA 1,0,7,213,221,94,0,
221,757
640 DATA 86,1,229,25,235,225,
26,254,1081
650 DATA 32,209,40,1,12,221,35
,221,771
660 DATA 35,5,242,107,112,126,
254,144,1028
670 DATA [illegible]
254,749
680 DATA 3,32,14,126,254,32,32
[illegible],304
690 DATA [illegible]
,40,748
700 DATA [illegible]
193,13,614
710 DATA 32,[illegible]
209,225,1109
720 DATA 237,[illegible]
,112,33,1117
730 DATA [illegible]
58,40,698
740 DATA [illegible]
48,119,862
750 DATA [illegible]
62,6,[illegible]
```

```
[illegible] DATA [illegible]
62,702
[illegible] DATA [illegible]
26,650
[illegible] DATA [illegible]
112,60,998
790 DATA 254,151,32,2,62,144,
90,251,946
800 DATA [illegible]
,218,40,1062
810 DATA [illegible]
110,48,127,1244
820 DATA 48,69,[illegible]
234,255,1177
830 DATA 1,0,24,0,23,0,22,0,70
840 DATA 255,255,232,255,0,0,0
[illegible]
```

Para entrar a colônia inicial, mova o cursor usando as teclas de controle (flechas). Pressione <ENTER> para criar uma célula na posição desejada, M para matar uma célula, e Q para terminar a entrada e iniciar o jogo.

```
10 CLS0:CLEAR 200,30999
20 FOR K=0 TO 11:T=0:FOR J=0 TO
 16:READ A:POKE 31000+K*17+J,A:
T=T+A:NEXT:READ B
30 IF T<>B THEN PRINT" ERRO NOS
 DADOS DA LINA";1000+K*10;"NAO
CORRA O PROGRAMA !!":END
40 NEXT
50 CLS 0:FOR K=0 TO 3:PRINT @K*
32,"geracao";:POKE 31203+K,[illegible]:N
EXT:POKE 65475,0
60 PMODE 3:POKE 179,128:PCLS
70 DEFUSR0=31000
80 X=2064:Y=1
90 P=PEEK(X):POKE X,[illegible] OR [illegible]*Y)A
ND(NOT(P AND 5*Y))
100 A$=INKEY$:IF A$="" THEN 90
110 POKE X,P
120 IF A$="^" AND X>1183 THEN X
=X-32
130 IF A$=CHR$(10) AND X<3040 T
HEN X=X+32
140 IF A$=CHR$(8) AND (X>1152 O
R(X=1151 AND Y=1)) THEN Y=Y+1:I
F Y>2 THEN Y=1:X=X-1
150 IF A$=CHR$(9) AND (X<3071 O
R(X=3071 AND Y=2)) THEN Y=Y-1:I
F Y<1 THEN Y=2:X=X+1
160 IF A$=CHR$(13) THEN 180
170 GOTO 90
180 N=USR0(0)
190 IF INKEY$[illegible] THEN 180
200 GOTO 50
1000 DATA 182,121,226,139,16,18
3,121,226,142,121,227,108,132,1
[illegible],128,120,50,2423
1010 DATA 37,9,134,[illegible],167,31,1[illegible]
0,121,231,37,239,198,4,206,4,11
,142,1759
1020 DATA 121,231,166,130,167,1
92,140,121,227,38,247,51,200,2[illegible]
,90,38,238,2425
1030 DATA 142,13,[illegible],204,[illegible],[illegible],237,
129,140,28,128,37,249,142,4,128
,206,1787
1040 DATA 13,64,166,128,133,10,
39,2,141,47,51,65,133,[illegible],39,2,14
[illegible],1179
1050 DATA 39,51,65,140,12,0,37,
233,142,4,128,206,13,64,166,132
,52,1484
1060 DATA 2,230,192,134,10,141,
32,230,192,134,5,141,46,53,2,16
7,128,1859
1070 DATA 140,12,0,37,231,57,52
,2,108,200,192,108,200,64,31,40
,196,1678
1080 DATA 63,39,8,108,200,191,1
08,95,108,200,63,193,63,39,8,10
8,200,1794
1090 DATA 193,108,63,108,200,66
,53,130,165,98,39,19,193,2,39,3
2,193,1702
1100 DATA [illegible],39,28,230,98,67,52,
2,228,224,231,98,32,17,193,3,38
,1583
1110 DATA 1[illegible],230,[illegible],196,143,250
,121,226,52,2,234,224,231,98,57
,0,0,2175
```

Para entrar a colônia inicial, movimente o cursor utilizando as teclas de controle (flechas). Pressione a barra de espaço para criar ou eliminar uma célula na posição desejada, e a tecla <ENTER> para terminar a entrada e iniciar o jogo.

Um detalhe importante: este programa não funcionará em um TRS-Color ou compatível que tenha um acionador de disquetes conectados, pois usa uma área de memória reservada.

[illegible] (TRS-Color)



É possível usar um sintetizador de voz para animar o jogo?

Você pode tornar o jogo ainda mais interessante programando seu micro para anunciar mensagens ou descrever o que se passa na tela.

Recorra ao manual do seu sintetizador para ver como introduzir as instruções que tornarão possível fazer a máquina falar.

Já publicamos em INPUT um artigo que examina em detalhe o uso de sintetizadores de voz. Consulte o

que examina em detalhe o uso de sin-
tetizadores de voz Consulte o

TABLETES GRÁFICOS

Desenhar na tela do computador usando centenas de comandos em BASIC não é um procedimento razoável. Com o auxílio de um tablete gráfico, você trabalhará tão facilmente quanto com lápis e papel.

Uma área de aplicação em que os computadores realmente excederam todas as expectativas é a do desenho auxiliado por computador, ou CAD (do inglês *Computer Aided Design*). Máquinas de todos os tamanhos, dos micros domésticos aos computadores de grande porte (*mainframes*), têm sido usadas nas mais diversas áreas do desenho e do projeto gráfico. Desenhistas de moda, por exemplo, ou empresas de construção civil, já empregam programas de CAD, e não apenas em projetos complexos.

Muitos micros domésticos foram planejados de modo a ter boa capacidade gráfica simplesmente para a implementação de jogos do tipo videogame. Desse ponto de partida, porém, desenvolveram-se programas de desenho gráfico com recursos espetaculares, sobretudo se levarmos em conta o tamanho e custo dos micros a que se destinam.

Um dos periféricos mais úteis para a exploração da capacidade gráfica dos computadores é o tablete gráfico, também chamado de *mesa digitalizadora*. O tipo mais comum permite que o usuário desenhe sobre uma superfície dura (o tablete), com o auxílio de uma caneta. O movimento da caneta no tablete é duplicado na tela do computador por meio de uma criativa combinação de hardware e software.

Para obter resultados equivalentes, sem empregar um tablete gráfico, seria preciso recorrer a um considerável volume de programação. Mesmo assim, não se conseguiria o mesmo grau de flexibilidade, sobretudo na alteração ou eliminação de partes do desenho.

Até os tabletes mais simples permitem o desenho de linhas sobre a tela. Dependendo do programa com o qual são usados, podem-se "pintar" áreas inteiras com uma certa cor, entre as disponíveis em seu micro. Os programas possibilitam ainda a mudança instantânea de cores no desenho, sob o comando de algumas teclas ou por seleção feita no próprio tablete. Efeitos de "pinceladas" de diferentes larguras também podem ser simulados pelo programa, em combinação com o tablete.

TIPOS DE TABLETE

Existem diversos tipos de tablete para microcomputadores pessoais. O mais conhecido no Brasil é o KoalaPad, destinado aos modelos da linha Apple. De baixo custo em relação aos demais, apresenta algumas limitações, como a baixa resolução gráfica.

Muitos outros modelos de tablete, como o DigiPad, o GrafPad (para o Spec-



trum) e o BitPad One (para diversos tipos de micro), podem ser encontrados no exterior. No Brasil, são mais comuns os tabletes digitalizadores para uso profissional, como os da marca Digigraf, bastante caros.

Os tabletes gráficos são vendidos em tamanhos que variam conforme a área de desenho disponível. Normalmente, suas medidas são especificadas segundo as dimensões padronizadas das folhas de papel (padrão DIN), tal como A3, A4, A6 etc. Medem, assim, de 100 × 100 mm (os menores, como o KoalaPad) a 250 × 300 mm (tabletes médios, usados com micros pessoais). Os modelos utilizados com mesas digitalizadoras de grande porte, para aplicações em engenharia, chegam a ter 1 000 × 750 mm. O tamanho, evidentemente, afeta o grau de precisão que se consegue atingir ao desenhar na superfície do tablete.

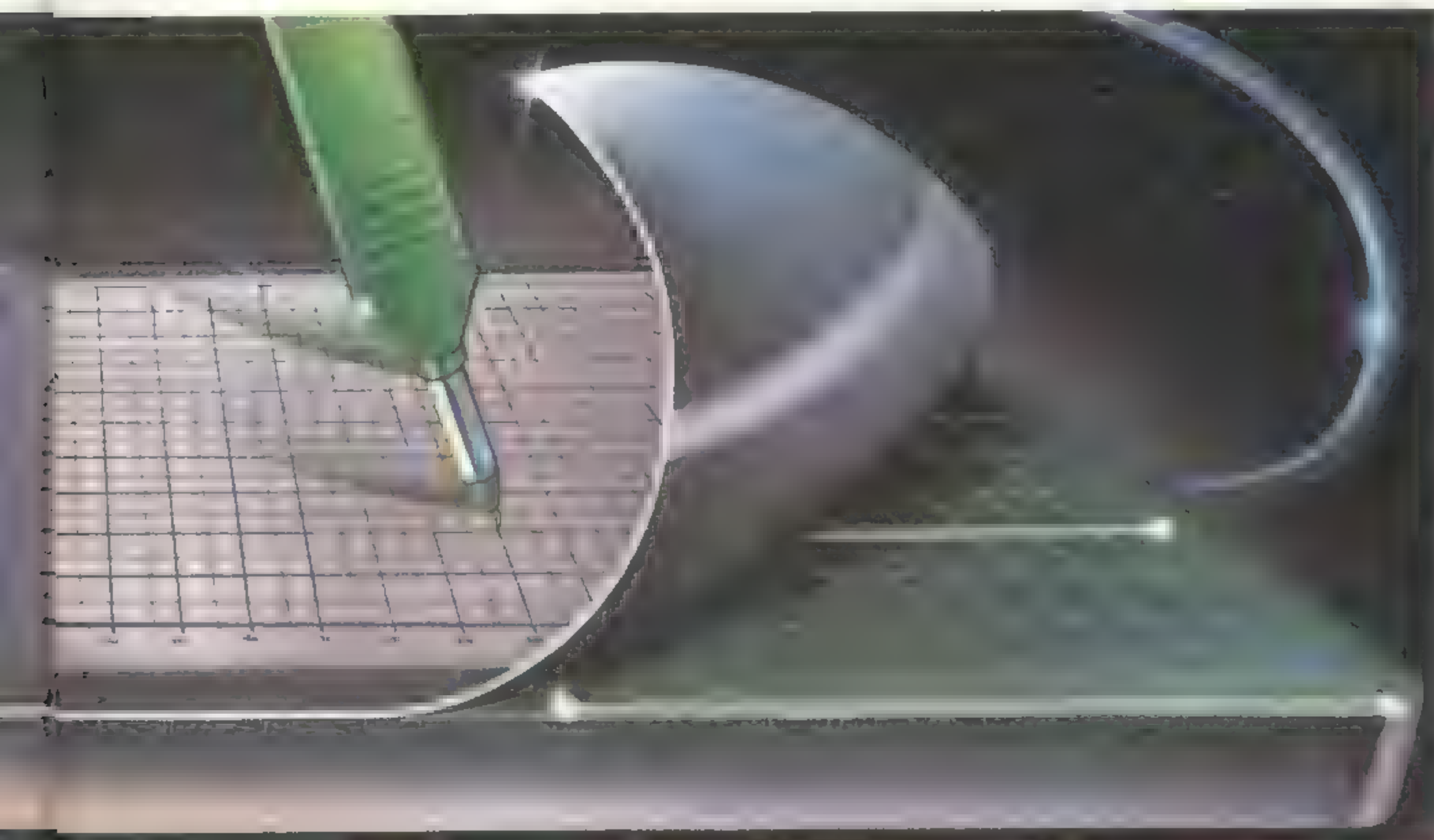
Os tabletes variam também conforme o cursor. Este pode ser um ponteiro ou uma caneta, preso ou não ao tablete por um fio, ou, então, uma "mira" formada por uma lente de vidro ou plástico, com uma cruz ao centro.

Existem ainda alguns tipos de digitalizadores gráficos que funcionam de modo diferente dos tabletes: são os *pantógrafos*, ou *traçadores de régua*. Esses dispositivos utilizam um braço articulado que se move em duas direções, X e Y. Na ponta do braço, prende-se uma caneta ou um lápis. O sistema pode ser mais barato do que um digitalizador de desempenho médio, mas, em contrapartida, sua precisão é menor. Além disso, não é fácil desenhar à mão livre com o pantógrafo. Em algumas situações, porém, esse dispositivo mostra-se de bastante utilidade — por exemplo, quando as coordenadas de pontos isolados precisam ser lidas rapidamente pelo computador.

COMO FUNCIONA

O tablete gráfico emprega o princípio da digitalização de curvas, amplamente usado em informática, já que muitas aplicações computacionais consistem na transformação adequada de um fenômeno analógico para uma seqüência de números binários (digitais).

Um sinal analógico varia continuamente e pode assumir qualquer valor fracionário, dentro de certos limites mínimo e máximo. Um exemplo de medidor analógico é o indicador de velocidade de um automóvel. Mas, como sabemos, o computador só é capaz de processar sinais digitais, ou seja, representados por meio dos números binários 0





e 1. A digitalização é justamente o processo de transformação de um sinal analógico em um sinal digital que o computador possa "entender" e armazenar.

O tablete gráfico, ou mesa digitalizadora, transforma desenhos em números (as coordenadas dos pontinhos que constituem as linhas do desenho). O computador, por sua vez, volta a transformar os números em desenhos, exibindo-os na tela com o máximo de fidelidade ao original. Sem a tradução de uma figura em números, o computador não poderia "entender" o desenho.

O processo de digitalização consiste em dividir uma curva no maior número possível de segmentos iguais, ou uma área no maior número possível de quadradinhos de igual tamanho. Quanto maior o número de partes, maior o detalhamento do desenho a ser entrado no computador. Contornos de figuras ou retas inclinadas em ziguezague são típicos de desenhos de baixa resolução.

O número e o tamanho dos quadrados de divisão da imagem dependem do tipo do tablete gráfico, assim como do computador. Em sistemas profissionais de alta resolução, o tamanho de um pixel chega a um centésimo de milímetro, e os contornos das figuras produzidas têm uma aparência contínua (como os desenhos animados gerados por computador). Os tabletes gráficos para micros não têm uma resolução tão alta, evidentemente, mas um décimo de milímetro já é suficiente para a produção de trabalhos de boa qualidade.

A área do tablete (bem como a do vídeo) é dividida em um quadriculado imaginário, cujo número de linhas e colunas varia de acordo com o computador empregado ou, ainda, com o grau de resolução gráfica (baixa, média e alta, nos micros das linhas Apple, MSX e TRS-Color) selecionado pelo programa ou pelo usuário.

É importante lembrar que a resolução máxima do tablete pode ser diferente da resolução gráfica da tela. Não tem sentido, por exemplo, utilizar um tablete com resolução de 100 000 pontos na área total, se a tela só representa 45 000 pontos. Estaremos desperdiçando recursos, a um certo preço, pois o computador não seria capaz de utilizar tanta informação gráfica. A situação contrária também ocorre — ou seja, pode-se utilizar um computador de alta resolução gráfica com um tablete de baixa resolução.

Muitos tabletes digitalizadores têm, no entanto, superfícies quadriculadas que permitem estabelecer uma correspondência com a representação na tela. As linhas e colunas podem ser numeradas, como se costuma fazer com a grade de referência de uma mapa.

Nesse tipo de grade, cada quadradinho tem duas coordenadas — o número da linha e o número da coluna — que correspondem, aproximadamente, às coordenadas X e Y do ponto do centro do quadradinho. Em um sistema onde o ponto de origem estivesse no canto inferior esquerdo do tablete, o quadradinho de número 10/23, por exemplo, estaria a uma distância de dez quadradinhos da esquerda, na horizontal, e a 23 quadradinhos de distância da borda inferior, na vertical. Isso significa, portanto, que um desenho — uma reta, por exemplo — pode ser convertido em uma série de números. Estes correspondem às coordenadas dos quadradinhos que a reta vai cruzar ao ser traçada sobre a superfície do tablete.

A GERAÇÃO DO SINAL

Existem diversas maneiras de produzir e enviar ao computador os pares de números correspondentes aos pontos da superfície do tablete.

Alguns tipos de mesa digitalizadora requerem uma caneta ou um apontador especial e dependem do contato direto da ponta desse dispositivo contra uma parte ativa da superfície; já outros exigem simplesmente o posicionamento próximo à superfície.

Entre as técnicas mais comuns de digitalização estão incluídas a conversão AD direta, a digitalização por grade de contato e a digitalização por grade de proximidade.

A conversão AD direta é usada apenas pelos digitalizadores do tipo pantográfico ou de régua. O método mais simples é o do digitalizador de régua. Este tem dois braços articulados: um que se move na direção horizontal, e outro, na vertical. Potenciômetros são conectados a cada braço, por meio de engrenagens ou de polias. À medida que deslocamos os braços pela superfície, os potenciômetros giram, produzindo uma voltagem contínua proporcional à distância percorrida naquela direção, desde a origem. Desse modo, obtemos as coordenadas X e Y dos pontos.

O digitalizador pantográfico, por sua vez, possui dois braços, articulados em um "ombro" e um "cotovelo", cada um dispondo de um potenciômetro. As coordenadas X e Y, nesse caso, têm que ser calculadas por métodos trigonométricos, a partir dos ângulos de posicionamento de cada braço.

Um conversor AD, ou analógico-digital, é responsável pela conversão da voltagem gerada nos potenciômetros. O conversor AD utiliza um processo de *comparação escalonada*: uma voltagem de referência, gerada internamente no conversor AD, é aumentada passo a passo, em pequenos incrementos fixos. A cada passo, ela vai sendo comparada com o sinal de entrada. Quando, finalmente, as duas voltagens se igualam, o número de passos que foram necessários é enviado ao computador. A voltagem de referência é então zerada, e o processo recomeça para um novo ponto do sinal de entrada. A *frequência de amostragem* corresponde ao número de vezes que isso é feito por segundo.

O digitalizador pantográfico requer dois conversores AD, um para cada potenciômetro. A resolução do conversor (ou precisão) é dada pelo número máximo de "partes" (valor do incremento) em que uma voltagem de entrada pode ser dividida, e está relacionada ao número de bits do conversor. Assim, um conversor de oito bits, por exemplo, pode gerar apenas um número entre 0 e 255 — ou seja, a voltagem máxima é dividida em 256 pedacinhos.

A informação gerada por cada conversor é transferida diretamente para a memória do computador, através de alguma porta de entrada (serial ou paralela). Se a freqüência de amostragem for muito grande, será necessário utilizar uma quantidade muito grande de memória. Por isso, a maioria dos programas gráficos não armazena toda a informação de um desenho em detalhes, mas simplesmente vai convertendo as coordenadas em linhas na tela.

SENSORES DE SUPERFÍCIE

O funcionamento dos tabletes existentes para micros baseia-se, geralmente, em dois tipos distintos de mecanismo. Ambos dependem de sensores ocultos sob a superfície do tablete.

Os tipos sensíveis ao contato direto são formados por duas folhas de plástico. Uma delas contém fios paralelos, espaçados na direção horizontal; a outra, fios espaçados na direção vertical. Ao se encostar a ponta do estilete ou da caneta na superfície do tablete, o contato elétrico estabelecido entre os fios horizontais e verticais, naquele ponto, informa ao computador as coordenadas do mesmo. Pode-se reconhecer facilmente esse tipo de tablete, já que qualquer instrumento pontiagudo, ou mesmo o próprio dedo, é capaz de ativar o sistema.

As mesas digitalizadoras mais precisas, entretanto, utilizam o princípio da proximidade por indução eletromagnética. A malha de fios embutida sob a superfície recebe correntes elétricas em um sistema de varredura constante. Na ponta da caneta ou do cursor especial existe uma bobina detetora, que não precisa estar em contato direto com a superfície. Pode-se, assim, colocar sobre esta uma folha de papel com o desenho já feito e copiá-lo.

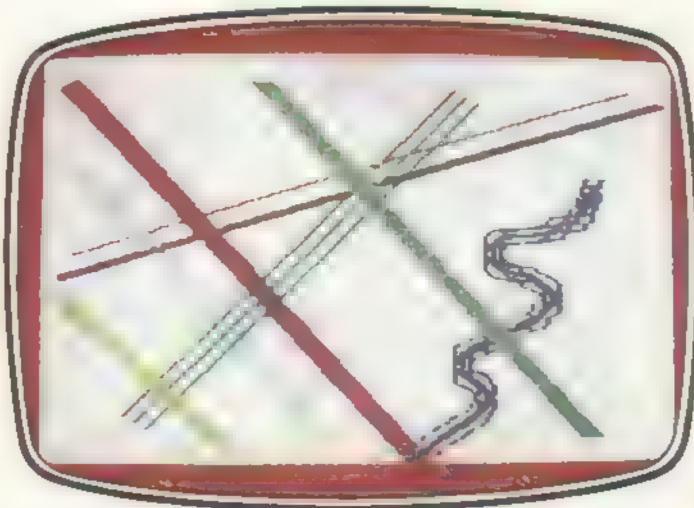
O SOFTWARE

Embora sejam muitas as diferenças dos tabletes digitalizadores quanto ao hardware, o que realmente conta, em termos da exploração dos recursos de cada um, é o software. Como este também é o responsável pela definição das características do desenho e pela facilidade com que ele pode ser executado, a maior preocupação do usuário deve ser a escolha do programa aplicativo. Para que não se cometam erros, convém sempre assistir a uma demonstração completa do sistema.

Evidentemente, o que o software é capaz de fazer depende muito do modelo do seu computador. Quanto mais rápido ele for, e quanto maior a sua memória, mais espaço existirá para a operação de um tablete sofisticado. Aplicações em alta resolução, assim como programas com recursos embutidos, exigem muita memória. É interessante, portan-



No menu de abertura, o KoalaPad mostra as opções disponíveis.



A largura das "pinceladas" pode ser escolhida à vontade.

to, utilizar disquetes, que facilitam o armazenamento de telas e expandem a capacidade do sistema, pelo uso do disco como memória virtual.

O mesmo acontece com a capacidade gráfica do sistema. Um bom programa para tabletes gráficos explora até o limite máximo os recursos disponíveis no microcomputador.

Muitos programas controlados apenas pelo teclado, como o publicado no artigo da página 414, são similares, em operação, ao software destinado aos tabletes. Porém, quanto à facilidade, rapidez, conveniência e flexibilidade, é enorme a diferença entre um sistema operado por teclado e o operado por um tablete. Um sistema controlado por teclado não permite, por exemplo, o desenho a mão livre.

COMO UTILIZAR UM TABLETE GRÁFICO

As combinações específicas entre computador/tablete variam bastante, pelas razões expostas anteriormente. Entretanto, todos os programas existentes foram projetados para trabalhar simulando o ato de desenhar ou pintar sobre uma folha de papel. É isto, mais do que qualquer outra coisa, que dá aos tabletes gráficos tanta vantagem em relação aos demais métodos de criação de imagens de alta resolução no microcomputador.

Em geral, um menu de abertura, como o existente para o KoalaPad, oferece ao usuário diversas opções quanto à técnica e tipo de desenho que se pretende executar. A escolha pode ser feita pelo teclado, ou, mais comumente, usando-se o próprio tablete gráfico como se fosse a paleta de um artista. A tela exibe potinhos de tinta, pincéis e outros materiais, que são selecionados com o cursor do tablete. Quando o cursor correspondente na tela estiver no lugar desejado, pressiona-se um botão ou tecla.

O desenho é executado na tela a mão livre, seja usando apenas a imaginação, seja seguindo um modelo feito sobre papel, que pode ser afixado ao tablete. Cabe ao programa receber os sinais gerados pelo tablete, interpretá-los e colocar instantaneamente a imagem traçada sobre a tela.

OS PROGRAMAS E SEUS RECURSOS

Algumas vezes, é difícil traçar uma reta, ou uma figura geométrica regular, a mão livre. Para isso, a maioria dos programas possui um menu de curvas e de figuras geométricas — como triângulos, retângulos, arcos, círculos, elipses, retas etc. — que podem ser escolhidas, posicionadas sobre a tela e traçadas automaticamente, no tamanho que se desejar.

Os programas também oferecem o recurso de apagar e corrigir partes do desenho, ou, nos sistemas mais sofisticados, de voltar atrás, desfazendo-se a última operação realizada. Se isso não for suficiente para a obtenção do resultado pretendido, pode-se ainda recorrer ao recurso de apagar toda a tela e começar tudo de novo.

É possível mudar a cor usada nos traços do desenho, assim como no preenchimento automático de áreas delimitadas (*paint*), a qualquer instante. Deve-se, inicialmente, delinear o desenho utilizando uma determinada cor. Em seguida, coloca-se o cursor gráfico no interior da área assim demarcada, e a seleção adequada é realizada. Se o espaço não estiver completamente fechado, a cor "escapa" para fora do mesmo, podendo tomar toda a tela. Por isso, convém sempre completar o desenho antes de começar a colori-lo.

Opções mais especializadas variam de programa para programa. Alguns possibilitam o traçado automático de imagens especulares (cópia invertida), ou a cópia de uma parte do desenho em outra posição. Esta operação é chamada de "carimbo", pois permite ao usuário criar, por exemplo, uma floresta inteira a partir do desenho de uma única árvore, ou, para maior variação, de sua cópia especular.

COMO GUARDAR UMA IMAGEM

Se você completou um desenho bonito e elaborado, certamente desejará armazená-lo permanentemente. Muitos softwares gráficos oferecem essa possibilidade por meio de um comando — dentro, é claro, das limitações impostas pelo hardware que você tem à disposição. As figuras armazenadas podem ser recuperadas depois, com um comando de leitura, e os recursos do programa permitirão, ainda, que você as modifique. Se quiser, combine-as com outras figuras, que podem ser obtidas da biblioteca de ícones, fornecidos junto com o sistema ou criados por você.

Armazene o desenho por tempo indeterminado ou utilize-o em outro programa — num jogo, por exemplo.

Para obter uma cópia da sua "obra de arte", fotografe a tela usando um filme em cores, ou copie o desenho em um *plotter* ou impressora gráfica.



A opção de "zoom" possibilita o desenho de pixels individuais.



Um recurso útil às composições é a repetição de elementos-chave.

AVALANCHE: A ROTINA PRINCIPAL

- TAREFAS DA ROTINA PRINCIPAL
- ORIGEM
- ACERTOS INICIAIS
- DANDO A PARTIDA

Já criamos uma página-título e uma página de instruções. Temos um tema musical e um cenário. Para completar o jogo, falta apenas uma rotina que controle as demais e dê vida a Willie.

Este é o sétimo artigo da série *Avalanche* e você deve estar pensando, com razão, que já é hora de colocar o programa em funcionamento.

Um jogo em linguagem de máquina, porém, precisa de uma rotina principal que chame as rotinas de instrução, de execução musical e de criação do cenário. Essa rotina deve ainda acertar os valores das variáveis do jogo, tais como o placar, o número de vidas e o nível de dificuldade. Sua origem será o endereço a ser chamado quando quisermos rodar o programa.

Depois de pronto, o programa será executado automaticamente assim que for lido da fita cassete. Por enquanto, contudo, não há razão para isso. Além do mais, precisamos ter acesso aos códigos do programa para terminá-lo, o que seria absolutamente impossível se sua execução fosse automática.



A listagem apresentada a seguir corresponde à rotina principal que dá início à versão do nosso jogo para os microcomputadores Spectrum.

```
 10 REM org 58576
 20 REM gbin call ti
 30 REM ld a,5
 40 REM ld (57343),a
 50 REM ld a,0
 60 REM ld (57344),a
 70 REM ld hl,0
 80 REM ld (57337),hl
 90 REM ld (57339),hl
100 REM ld (57341),hl
110 REM nlv ld a,19
120 REM ld (msk+1),a
130 REM ret
140 REM org 58035
150 REM ti *
160 REM org 58281
170 REM msk *
```

A origem é o endereço que chamaremos por intermédio do comando **RANDOM USR** para que o programa seja executado. Não se esqueça de anotá-lo.

Os rótulos que fazem parte da listagem não são chamados por esta seção do jogo. Eles serão utilizados por rotinas futuras, quando nosso personagem tiver morrido e você quiser começar o jogo novamente.

ENDEREÇOS

O programa começa chamando a sub-rotina ti, que imprime os títulos e a página de instrução.

Vários parâmetros relativos à contagem de pontos são então estabelecidos. O escore propriamente dito é colocado nos endereços 57337 a 57342. O número de vidas fica em 57343 e o nível de dificuldade em 57344.

ACERTOS INICIAIS

Para que o jogador tenha direito a cinco vidas, o número 5 é colocado em 57343, com o auxílio do acumulador. Zero é colocado do mesmo modo em 57344, fazendo com que o nível de dificuldade inicial seja 0.

O escore inicial, zero, é acertado colocando-se 0 nos endereços 57337, 57338, 57339, 57340, 57341 e 57342, por meio do par de registros HL.

Em seguida, A recebe o valor 19, e a rotina musical **msk** é ajustada para executar as dezenove primeiras notas da melodia *Greensleeves*.



Esta é a rotina que dá início à versão do nosso jogo para o TRS-Color.

```
10  ORG 19426
20  GBIN JSR START
30  LDA #5
40  STA 18239
50  CLR 18238
60  LDX #18240
70  LDB #6
80  GBINI CLR ,X+
90  DECB
100  BNE GBINI
110  RTS
120  START EQU 19000
```

A origem corresponde ao endereço a ser chamado por intermédio da instrução **EXEC**, quando quisermos iniciar a execução do programa. Não se esqueça de anotar esse número.

Os rótulos da listagem não são utilizados nesta parte da rotina. Eles serão chamados por outras seções do programa, que publicaremos mais tarde. Sua função é recomeçar o jogo, caso você queira prosseguir quando Willie for fatalmente vitimado.

DANDO A PARTIDA

Antes de mais nada, o programa chama a sub-rotina **START**, responsável pela impressão da página-título e da página de instruções, que abrem o jogo. Em seguida, são estabelecidos diversos valores iniciais que influem na contagem final de pontos.

O nível de dificuldade do jogo é armazenado no endereço de memória 18238. O número de vidas, por sua vez, fica em 18239. O escore propriamente dito é guardado em seis bytes, a partir do endereço 18240.

PRIMEIROS ACERTOS

O número inicial de vidas de Willie, 5, é colocado no acumulador e transferido para 18239. A instrução **CLR** limpa a posição de memória 18238, para zerar o nível de dificuldade.

O registro X recebe o endereço inicial dos bytes de escore e o contador B recebe o numero 6 — número de bytes de escore. A instrução **CLR ,X+** limpa o conteúdo da posição de memória apontada por X e soma uma unidade ao valor de X. **DEC B** diminui B em uma unidade. A instrução **BNE** retorna ao rótulo **GBINI** para apagar o próximo byte até que B seja zero. Esse pequeno laço limpa todas as posições de 18240 a 18245 — o que equivale a zerar todos os endereços que se referem ao escore, um depois do outro.



Esta é a rotina principal que dá início ao jogo na versão do MSX.

```
org -11670
gin call -12288
call -12166
ld a,5
ld (-5221),a
ld a,0
ld (-5228),a
ld hl,0
ld (-5219),hl
ld (-5217),hl
ld (-5215),hl
hlv ld a,19
ld (-12162),a
ret
end
```

A origem dessa rotina é o endereço que você chamará por intermédio de dois comandos, **DEFUSR** e **USR(0)**, para começar a execução do jogo. Não se esqueça de anotar esse número.

Os rótulos que precedem as instruções não são chamados nesta parte do programa. Serão utilizados pelas rotinas que inicializam o jogo quando o pobre Willie tiver sido morto e você quiser jogar de novo.

DANDO A PARTIDA

Inicialmente a rotina chama as sub-rotinas -12228 e -12166. Enquanto a primeira imprime o título e a página de instruções, à segunda cabe executar a música *Greensleeves.*

Os vários parâmetros da contagem de pontos são, então, inicializados. O escore propriamente dito é colocado nos endereços -5219 até -5214. O número de vidas fica em -5221 e o nível de dificuldade, em -5228.

ACERTOS INICIAIS

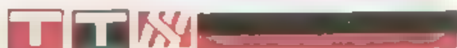
O número 5 é colocado em -5221, por meio do acumulador, para que o jogador inicie a partida dispondo de cinco vidas. Do mesmo modo, 0 é colocado em -5228, para que o nível de dificuldade inicial seja igual a 1.

Os endereços -5219, -5218, -5217, -5216, -5215 e -5214, por sua vez, são preenchidos com o valor 0, por intermédio do par de registros HL, zerando o escore inicial.

Em seguida, a rotina responsável pela música — colocada no endereço -12166 — é ajustada para executar exclusivamente as dezenove primeiras notas da melodia *Greensleeves.*

ROTINA EM CÓDIGO DE MÁQUINA (1)

Programas sofisticados e rápidos exigem a combinação de código de máquina e BASIC. Aprenda aqui uma série de truques para tornar mais eficaz a interação entre ambos.



As mais modernas versões do interpretador BASIC desenvolvido pela empresa norte-americana Microsoft foram implementadas nos microcomputadores das linhas TRS-80, TRS-Color, MSX e IBM-PC. Tanto elas quanto o MBASIC (usado em qualquer micro que tenha o sistema operacional compatível com o CP/M) possuem diversos recursos para dar acesso direto às posições absolutas de memória do computador, bem como à execução de programas em código de máquina a partir de um programa em BASIC.

Já estudamos os principais elementos do BASIC para efetuar essas funções: o **PEEK** e o **POKE**. Porém, para uma interação mais eficaz entre programas em linguagem de máquina e programas em BASIC, outros comandos são necessários. Destacam-se, entre eles, o **DEFUSR**, o **USR** e o **VARPTR**.

Em artigos anteriores, vimos diversos exemplos de como executar um programa em linguagem de máquina a partir de um programa em BASIC. O método mais comum consiste em armazenar o programa em código de máquina em uma parte não usada da memória RAM (ou seja, em uma parte não sujeita a ser invadida pelo programa BASIC ou por sua área de variáveis), e executá-lo por meio de um comando **USR** colocado em um ponto qualquer do programa em BASIC. O comando **DEFUSR**, ou equivalente, é usado para indicar a locação inicial do programa em linguagem de máquina.

Esse método, entretanto, apresenta alguns problemas:

- O programa em código de máquina precisa ser montado à parte e armazenado em fita ou disco, para, então, ser carregado na memória.
- É necessário reservar uma parte protegida da memória para armazenar o programa, o que pode ser inconveniente em muitos tipos de aplicação.

A FUNÇÃO VARPTR

Examinaremos, neste artigo, uma técnica alternativa, que consiste na utilização da função **VARPTR** para armazenar programas em linguagem de máquina dentro do próprio programa em BASIC. Essa técnica proporciona poderosos recursos de programação para os usuários das linhas TRS-80, TRS-Color e MSX (os micros das linhas Sinclair e Apple dispõem do comando **CALL**, equivalente ao **USR**, mas não têm a função **VARPTR**). Veremos aqui como empregar a função **VARPTR** em micros da linha TRS-80; em artigos futuros trataremos do TRS-Color e do MSX.

O "truque" é muito simples: em vez de armazenarmos um programa em código de máquina numa parte reservada de memória, nós o "enxertamos" dentro do programa em BASIC que o utilizará. Assim, quando o programa em BASIC for carregado da fita ou disco, para ser executado, trará junto, automaticamente, o programa ou programas em linguagem de máquina. Esse método permite a definição e armazenamento do número que quisermos de rotinas em linguagem de máquina, cada qual bem identificada dentro do programa em BASIC.

Mas onde armazenar código de máquina dentro de um programa em BASIC? Existem diversas técnicas disponíveis. As mais flexíveis, entretanto, usam o seguinte recurso: uma constante alfanumérica (por exemplo, **PG$**) é definida dentro do programa, com um comprimento correspondente ao número de bytes do programa em código de máquina. Suponhamos, por exemplo, que a rotina em código de máquina terá doze bytes de extensão. Definimos, então:

```
20 PG$="            "
```

Observe que foram colocados doze espaços em branco entre as aspas da linha 20. No lugar desses espaços, porém, poderiam introduzir quaisquer caracteres, iguais ou diferentes. O que vale é o número correto de caracteres. Muitos programadores preferem usar uma seqüência numérica para assegurar que o número de bytes seja correto:

```
20 PG$="123456789012"
```

Feito isso, carregamos os bytes disponíveis (que estão automaticamente fixos e reservados na área de memória de programa) na variável **PG$**, com os códigos decimais, binários, octais, ou hexadecimais correspondentes ao programa em código de máquina.

Um pequeno programa, contendo um laço **FOR...NEXT**, pode se encarregar dessa tarefa. Ele lerá os códigos, armazenados em linhas **DATA**, e os colocará nas locações absolutas de memória, correspondentes à variável **PG$**, usando comandos **POKE**.

A operação só precisa ser feita uma vez. Depois, o programa de carregamento pode ser apagado.

O processo é simples, mas há um detalhe técnico a ser resolvido: a localização das posições da memória onde **PG$** está armazenada. Teoricamente, podemos fazer esse cálculo para qualquer programa, desde que saibamos a locação absoluta da memória onde ele começa (geralmente fixa para cada tipo de computador), e quantos bytes gasta cada linha do programa até chegarmos à linha em que **PG$** é definida.

Na prática, porém, tal cálculo é difícil e demorado, podendo tornar-se realmente trabalhoso se o programa ainda estiver em desenvolvimento. Nesse caso, cada alteração no programa, feita acima da linha onde a variável **PG$** é definida, altera a sua locação na memória.

ONDE ESTÃO AS VARIÁVEIS?

Felizmente, é possível encarregar o próprio programa de calcular automaticamente a localização de **PG$** na memória. Para isso, devemos usar a função **VARPTR**, cujo nome deriva da expressão inglesa *VARiable PoinTeR*, ou apontador de variável.

Para entender como essa função trabalha, precisamos saber como o computador armazena cadeias alfanuméricas na memória. Como vimos no artigo da página 947, seu programa pode ter acesso direto a toda essa informação por meio da função VARPTR.

Recapitulando: toda vez que uma variável alfanumérica é definida, o interpretador BASIC cria três bytes *apontadores*, que contêm o comprimento da cadeia alfanumérica e o endereço absoluto de onde ela se inicia. Por exemplo, para uma variável **A$**:

■ O USO DE ROTINAS EM CÓDIGO DE MÁQUINA DENTRO DE PROGRAMAS BASIC
■ OS COMANDOS USR E DEFUSR
■ COMO ARMAZENAR ROTINAS USR EM UM PROGRAMA
■ DIFERENÇAS ENTRE O BASIC PARA CASSETE E PARA DISCO
■ UM PROGRAMA PARA PREENCHER A TELA

VARPTR(A$) = locação do apontador de **A$**
PEEK(VARPTR(A$)) = comprimento de **A$**
PEEK(VARPTR(A$)+1) – byte menos significativo do endereço de **A$**
PEEK(VARPTR(A$)+2) – byte mais significativo do endereço de **A$**

UM PROGRAMA COMPLETO

Sabendo como determinar o endereço inicial de uma variável *string* na memória, fica fácil escrever um programa para carregar código de máquina na mesma. Vejamos, por meio de um exemplo, como isso funciona.

A rotina em linguagem de máquina, apresentada a seguir, preenche toda a tela com um caractere qualquer, previamente colocado na primeira posição da tela (por exemplo, se colocarmos * na posição 0 da tela, uma chamada à rotina preencherá a tela com asteriscos quase que instantaneamente).

```
10 ORG 0BFF0H      ;ORIGEM
20 LD HL,15360     ;HL APONTA P/0
30 LD DE,15361     ;DE APONTA P/1
40 LD BC,1023      ;POR 1023 VEZES
50 LDIR            ;REPETE
60 RET             ;VOLTA AO BASIC
70 END
```

Quando traduzido por um Assembler, esse curto programa daria doze bytes em hexadecimal:

Linha 20:	21	00	3C
Linha 30:	11	01	3C
Linha 40:	01	FF	03
Linha 50:	ED	B0	
Linha 60:	C9		

que, em decimal, correspondem a:

```
33,0,60,17,1,60,1,255,3,237,176
,201
```

Vamos escrever agora nosso programa principal — ou seja, o programa em linguagem BASIC que utiliza a rotina em código de máquina.

Não rode o programa ainda, pois o código de máquina não foi carregado:

```
10 CLS:PRINT"PROGRAMA DE TESTE
20 PG$="            "
30 D=PEEK(VARPTR(PG$)+1) + 256
* PEEK(VARPTR(PG$)+2)
40 DEFUSR0=D
80 PRINT:INPUT "CARACTERE ";C$
90 CLS:PRINT C$
100 X=USR0(0)
110 FOR I=1 TO 1000:NEXT I
120 GOTO 10
```

O programa funciona da seguinte maneira: a linha 20 define um *string* fixo, **PG$**, na memória de programa, com o comprimento de doze bytes.

As linhas 30 e 40 localizam o endereço da variável **PG$** e o definem como início de uma rotina de máquina.

O comando usado para isso denomina-se **DEFUSR**, que é uma abreviatura da expressão *DEFine USeR function*. Seu objetivo é informar ao programa principal o endereço absoluto inicial da rotina. Nos microcomputadores da linha TRS-80 com BASIC para disco, podem-se definir até dez rotinas de usuário simultaneamente, por meio dos comandos **DEFUSR0** a **DEFUSR9**, que corresponderão às chamadas **USR0** a **USR9**.

Para os microcomputadores da linha TRS-80 com BASIC para cassete, pode-se definir apenas uma rotina **USR**. Nesse caso, não se pode contar com o comando **DEFUSR** para informar o seu endereço de partida. Temos, assim, que dar dois **POKE** em um apontador específico da memória de trabalho do computador. Eles correspondem aos bytes de endereço 16526 (para o byte menos significativo) e 16527 (para o byte mais significativo). Esse processo torna possível copiar diretamente os bytes obtidos pela função **VARPTR**. Para computadores com BASIC para cassete, portanto, devemos substituir as seguintes linhas no programa anterior:

```
30 POKE 16526,PEEK(VARPTR(
PG$)+1)
40 POKE 15527,PEEK(VARPTR(
PG$)+2)
```

Como a variável **PG$** está fixa dentro do programa, a definição com **DEFUSR** ou com **POKE** só precisa ser realizada uma vez.

A linha 80 pede ao usuário que entre pelo teclado o caractere com que deseja preencher a tela. A linha 90 limpa a mesma e coloca o caractere indicado na primeira posição.

Finalmente, a linha 100 executa a rotina, por meio da função fictícia **USR** (da expressão, em inglês, *USeR function*). Como não se trata de um comando, **USR** pode ser colocada dentro de uma expressão matemática válida. No exemplo, não precisamos passar nenhum argumento para a rotina de máquina (o que seria feito por meio do valor entre parênteses), nem obter um valor de retorno (o que seria feito através do próprio valor da função **USR**). Como vimos anteriormente, nos microcomputadores com BASIC para disco, cada **DEFUSRn** (onde **n** é um número inteiro entre 0 e 9) corresponde à rotina **USRn**

Como os computadores com linguagem BASIC para cassete aceitam apenas uma rotina de usuário, devemos substituir a linha 100 por:

```
100 X=USR(0)
```

PROGRAMA DE CARREGAMENTO

Antes de usar ou gravar o programa principal, é necessário carregar o programa de código de máquina na variável **PG$**. Para fazê-lo, anexe ao programa anterior as seguintes linhas:

```
50 FOR I%=D TO D+12:READ N% :
POKE I%,N% : NEXT I%
60 DATA 33, 0, 60, 17, 1, 60,
1, 255, 3, 237, 176, 201
70 STOP
```

O programa pode ser apagado com:

```
DELETE 50-70
```

Você notará que, ao listar o programa principal, a linha 30 não estará mais vazia: ela conterá códigos gráficos e até mesmo comandos em linguagem BASIC, dependendo da linha do computador que você estiver usando.

Você pode agora gravar e executar o programa quantas vezes quiser, sem ter que carregar separadamente o programa em linguagem de máquina. E, é claro, até dez variáveis alfanuméricas diferentes poderão ser usadas no mesmo programa. Elas definirão rotinas de máquina separadas, que serão chamadas de **USR0**, **USR1**, **USR2** etc.

A ARANHA MARCIANA (2)

O jogo está completo. Armado com suas flechas, Freddy observa os balões que começam a subir. Precisará de muito sangue frio para estourá-los e impedir que a aranha venha em sua direção.

Já digitamos as rotinas encarregadas de inicializar o jogo e de fazer a montagem dos gráficos.

Vamos agora completar o programa, adicionando as rotinas de animação.

O LAÇO PRINCIPAL

```
10 CLEAR 65287
20 GOSUB 1000
30 GOSUB 3000
50 IF ax<>29 THEN  GOSUB 300
70 GOSUB 400
90 GOSUB 500
100 GOSUB 200  IF dead=0 THEN
GOTO 50
```

```
10 CLS 3: PRINT @266,"INICIALIZ
ANDO"; SCREEN 0,1
20 GOSUB 1000
25 GOSUB 1600
30 GOSUB 3000
50 IF AX<>29 THEN GOSUB 300
70 GOSUB 400
90 GOSUB 500
100 GOSUB 210:IF DD=0 THEN 50
```

```
10 CLS:COLOR 1,14,14
15 OPEN "GRP:" FOR OUTPUT AS #1
20 GOSUB 1000
30 GOSUB 3000
50 IF AX<>29 THEN GOSUB 300
70 GOSUB 400
90 GOSUB 500
100 GOSUB 210:IF DD=0 THEN 50
```



```
10  TEXT : HOME : PRINT "UM MOM
ENTO..."
15  HIMEM: 18817
20  GOSUB 1000
30  GOSUB 3000
50  IF AX < > 28 THEN  GOSUB 3
00
70  GOSUB 400
90  GOSUB 500
100  GOSUB 210: IF DD = 0 THEN
50
```

Esse programa, o principal do jogo, foi estruturado de modo a chamar apenas as sub-rotinas necessárias. Primeiramente, ele define a tela e reserva espaço na parte superior da memória (Apple, TK-2000 e Spectrum). As sub-rotinas chamadas nas linhas 20 a 30 (listadas no artigo anterior) têm a função de definir os blocos gráficos e inicializar o valor do recorde.

O laço principal do programa vai da linha 50 à linha 100, e se repete enquanto Freddy estiver vivo (variável **dead** ou **DD** maior do que zero). Esse laço é responsável pelo movimento da flecha, da aranha, de Freddy e dos balões. Cada rotina chamada atualiza as variáveis dentro do laço.

HORA DO "ALMOÇO"



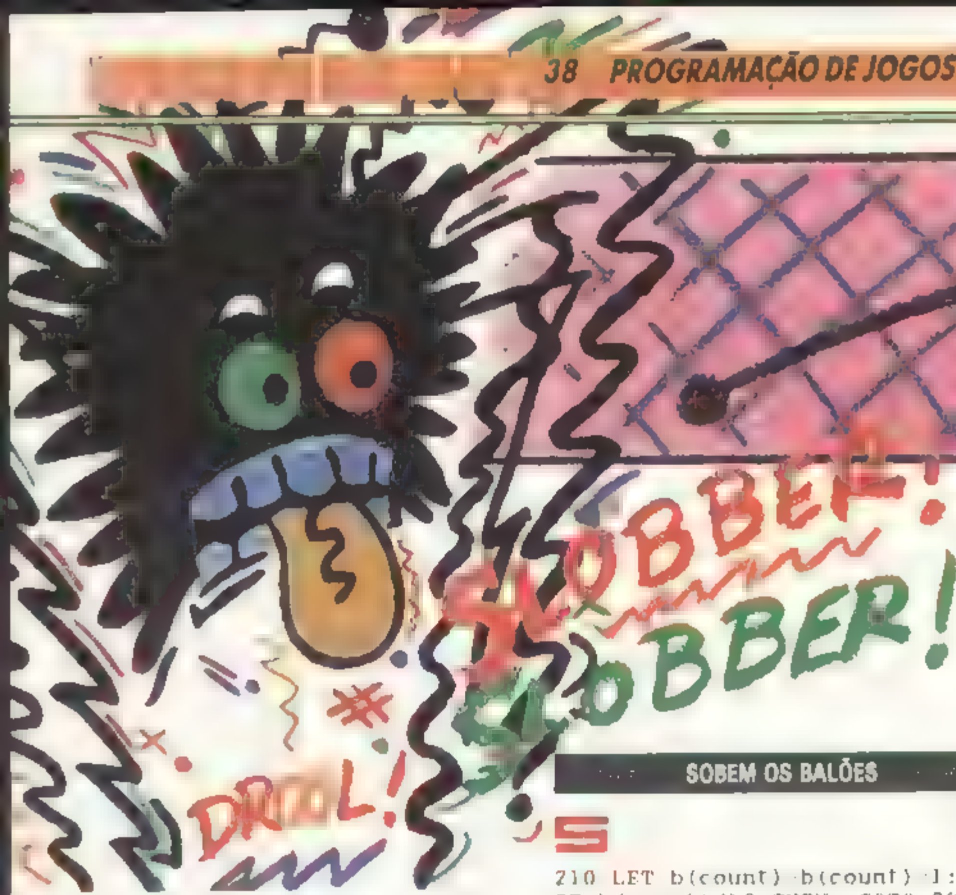
```
105 LET s(xinc)=1
110 FOR x=s(xpos) TO 29: GOSUB
500: NEXT x
120 LET s(yinc)=1: LET s(xinc)
```

```
=0
125 FOR y=s(ypos) TO 19
130 IF y=my AND ax=29 THEN
POKE 23607,60  PRINT AT my+1,
29," ": POKE 23697,252
140 GOSUB 500
150 NEXT y
160 POKE 23607,60: PRINT AT 10
,0; INK 2, PAPER 7; BRIGHT 1;
FLASH 1,"Voce esta morto!  Out
ra jogada ?",  POKE 23607,252
165 LET a$=INKEY$  IF a$=""
THEN  GOTO 160
170 IF a$="s" OR a$="S" THEN
GOTO 30
175 IF a$<>"n" AND a$<>"N"
THEN  GOTO 160
180 POKE 23607,60  CLS   STOP
```



```
105 S(XI)=1
110 FOR X=S(XP) TO 29:GOSUB 500
:NEXT X
120 S(YI)=1:S(XI)=0
125 FOR Y=S(YP) TO 19
130 IF Y=MY AND AX=29 THEN PUT(
232,(MY+1)*8)-(239,(MY+1)*8+7),
S1,PSET
140 GOSUB 500
150 NEXT Y
160 FOR SL=180 TO 160 STEP -1:S
OUND SL,1:NEXT:CLS:PRINT @256,"
 VOCE ESTA MORTO !
  QUER RECOMECAR  (S/N) ?"
165 A$=INKEY$:IF A$="" THEN 165
170 IF A$="S" THEN 30
175 IF A$<>"N" THEN 160
180 CLS:END
```

```
105 S(XI)=1
110 FOR X=S(XP) TO 29:GOSUB 500
:NEXT X
120 S(YI)=1:S(XI)=0
125 FOR Y=S(YP) TO 19
130 IF Y=MY THEN PUT SPRITE 2,(
X2,Y2),0,FD:PUT SPRITE 3,(X2,Y2
),0,FL
140 GOSUB 500
150 NEXT Y
160 PLAY"ABCDEF":FOR X=1 TO 300
:NEXT:SCREEN0:CLS:PRINT"Você es
tá morto!":LOCATE 0,12:PRINT"Jo
ga novamente? (S/N)"
165 A$=INKEY$:IF A$="" THEN 165
170 IF A$="S" THEN RUN
175 IF A$<>"N" THEN 160
180 CLS:END
```



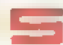


```
105 S(XI) = 1
110  FOR X = S(XP) TO 29: GOSUB
 500  NEXT X
120 S(YI) = 1:S(XI) = 0
125  FOR Y = S(YP) TO 19
130  HCOLOR= 0: FOR X = 16 TO 2
4: HPLOT 230,Y * 8 + X TO 260,Y
 * 8 + X: NEXT : HCOLOR= 3
140  GOSUB 500
150  NEXT Y
160  FOR SL = 1 TO 5: PRINT  CH
R$ (7);: NEXT : TEXT : HOME : P
RINT "VOCE ESTA MORTO!": VTAB 1
5: PRINT "QUER RECOMECAR? (S/N)
";
170  GET A$
180  IF A$ = "S" THEN  HOME : G
OTO 30
190  HOME : END
```

Após a remoção de todas as portas da gaiola da aranha, a variável **dead** (ou **DD**, nos outros microcomputadores) assume o valor 1 e as linhas 105 a 180 são executadas (190 no Apple e no TK-2000).

A aranha move-se horizontalmente até chegar logo acima de Freddy. Em seguida, realiza um movimento vertical, alcançando nosso infeliz personagem, que é, então, devorado. Quando isso ocorre, o jogador tem a opção de jogar mais uma vez (linhas 160 a 180). No programa do Spectrum, deve-se colocar o valor 60 na posição 23607 para que todo o conjunto de caracteres seja usado novamente.

SOBEM OS BALÕES



```
210 LET b(count)=b(count)-1:
IF b(count)<>0 THEN  GOTO 280
220 LET b(count)=b(maxcount):
PRINT AT b(ypos)+1,b(xpos);"
";: LET b(ypos)=b(ypos)-1: IF
b(ypos)=4 THEN  GOSUB 600:
POKE 23607,60: PRINT AT 1,10+(
3-props)*9;"";AT 2,10+(3-props
)*9;" ": POKE 23607,252: LET
props=props-1
225 IF props=0 THEN  LET dead=
1
230 IF ((ay<>b(ypos) AND ay<>b
(ypos)+1) OR (ax<b(xpos)-1 OR
ax>b(xpos)+1)) THEN  GOTO 250
240 LET score=score+b(points):
GOSUB 600: IF score>hiscore
THEN  LET hiscore=score: POKE
23607,60: PRINT AT 0,23; INK 0
; PAPER 6;hiscore: POKE 23607,
252
245 GOTO 380
250 GOSUB 4300
280 RETURN
```



```
210 B(CT)=B(CT)-1:IF B(CT)<>0 T
HEN 280
220 B(CT)=B(MC)
221 B(YP)=B(YP)-1: IF B(YP)=4 T
HEN X2=B(XP)*8:Y2=B(YP)*8+8:PUT
(X2,Y2)-(X2+15,Y2+15),SP,PSET:G
OSUB 600:X2=(10+(3-PP)*9)*8:PUT
 (X2,8)-(X2+7,15),S1,PSET:PUT(X
2,16)-(X2+7,23),S1,PSET:PP=PP-1
225 IF PP=0 THEN DD=1
230 IF ((AY<>B(YP) AND AY<>B(YP
)+1) OR (AX<B(XP)-1 OR AX>B(XP)
+1)) THEN 250
240 SC=SC+B(PO):X2=B(XP)*8:Y2=(
B(YP)+1)*8:PUT(X2,Y2)-(X2+15,Y2
+15),SP,PSET:GOSUB 600:IF SC>HS
 THEN HS=SC:GOSUB 1700
245 GOTO 400
250 GOSUB 4300
280 RETURN
```



```
210 B(CT)=B(CT)-1:IF B(CT)<>0 T
HEN 280
220 B(CT)=B(MC)
222 B(YP)=B(YP)-1:IF B(YP)=4 TH
EN GOSUB 600:LINE ((10+((3-PP)*
9))*8,7)-((10+((3-PP)*9))*8+7,2
4),14,BF:PP=PP-1
225 IF PP=0 THEN DD=1
230 IF ((AY<>B(YP) AND AY<>B(YP
)+1) OR (AX<B(XP)-1 OR AX>B(XP)
+1)) THEN 250
240 SC=SC+B(PO):GOSUB 600:IF SC
>HS THEN HS=SC:GOSUB 1700
245 GOTO 400
250 GOSUB 4300
280 RETURN
```



```
210 B(CT) = B(CT) - 1: IF B(CT)
 < > 0 THEN 280
220 B(CT) = B(MC)
222 B(YP) = B(YP) - 1: IF B(YP)
 = 4 THEN X2 = B(XP) * 8:Y2 = B
(YP) * 8 + 8: HCOLOR= 0: DRAW B
A AT X2,Y2: GOSUB 600: HCOLOR=
0: FOR X = 0 TO 6: HPLOT X + (4
 - PP) * 77,4 TO X + (4 - PP) *
 75,22: NEXT : PP = PP - 1
225  IF PP = 0 THEN DD = 1
230  IF ((AY < > B(YP) AND AY
< > B(YP) + 1) OR (AX < B(XP)
- 1 OR AX > B(XP) + 1)) THEN 25
0
240 SC = SC + B(PO):X2 = B(XP)
* 8:Y2 = (B(YP) + 1) * 8: HCOLO
R= 0: DRAW BA AT X2,Y2: HCOLOR=
 3: GOSUB 600: IF SC > HS THEN
HS = SC: GOSUB 1700
245  GOTO 400
250  GOSUB 4300
280  RETURN
```

Os elementos mais importantes da matriz do balão são **b (count)** e **b (maxcount)**, para o Spectrum, e **B (CT)** e **B (MC)**, para as outras linhas. Cada vez que a rotina é acionada, a linha 210 decrementa **b (count)** ou **B (CT)**, quando ela atinge 0, o balão se move. Em segui-



da, a linha 220 copia o número de **b (maxcount)** para **b (count)** – ou **B (MC)** para **B (CT)**, nos demais computadores. A velocidade do balão é determinada pelo valor de **b (maxcount)** — ou **B (MC)**. A linha 220 verifica se o balão foi atingido ou se chegou ao topo da tela.

Se o balão foi atingido, o placar é incrementado; se ele chegou ao topo, uma porta é removida. Quando todas as portas forem removidas, a variável **dead** (ou **DD**) assume o valor 1.



O ESTOURO DO BALÃO



```
300 PRINT AT ay,ax;"  "· LET
ax=ax-1: IF ax<0 THEN  LET ax=
29: PRINT AT my+1,29;"e": LET
ay=my+1· RETURN
310 IF ((ay=b(ypos) OR ay=b(yp
os)+1) AND (ax=b(xpos) OR ax=b
(xpos)+1)) THEN  LET score=sco
re+b(points). GOSUB 600: IF sc
ore>hiscore THEN  LET hiscore=
score: POKE 23607,60: PRINT AT
 0.23; INK 0; PAPER 6;hiscore:
 POKE 23607,252
330 IF ax<>29 THEN  GOSUB 4100
340 RETURN
```

```
300 PUT(AX*8,AY*8)-(AX*8+15,AY*
8+7),S2,PSET:AX=AX-1:IF AX<0 TH
EN AX=29:PUT(232,(MY+1)*8)-(239
,(MY+1)*8+7),E,PSET:AY=MY+1:RET
URN
310 IF ((AY=B(YP) OR AY=B(YP)+1
) AND (AX=B(XP) OR AX=B(XP)+1))
 THEN SC=SC+B(PO):X2=B(XP)*8:Y2
=B(YP)*8+8:PUT(X2,Y2)-(X2+15,Y2
+15),SP,PSET:GOSUB 600:IF SC>HS
 THEN HS=SC:GOSUB 1700
330 IF AX<>29 THEN GOSUB 4110
340 RETURN
```

```
300 AX=AX-1 IF AX<0 THEN AX=29·
PUT SPRITE 3,(232,(MY)*8),4,FL·
AY=MY·RETURN
310 IF ((AY=B(YP) OR AY B(YP)+1
) AND (AX=B(XP) OR AX=B(XP)+1))
 THEN SC=SC+B(PO) GOSUB 600 IF
SC>HS THEN HS=SC·GOSUB .700
330 IF AX<>29 THEN GOSUB 4.10
340 RETURN
```

```
300  HCOLOR= 0· DRAW FL AT AX *
 8,AY * 8· HCOLOR= 3 AX = AX
1: IF AX < 0 THEN AX = 28: DRAW
 FL AT AX * 8,(MY + 1) * 8:AY =
 MY + 1: RETURN
310  IF ((AY = B(XP) OR AY   B(
YP) + 1) AND (AX = B(XP) OR AX
= B(XP) + 1)) THEN SC = SC + B(
PO)·X2 = B(XP) * 8·Y2 = B(YP) *
 8· HCOLOR= 0· DRAW BA AT X2,Y2
· HCOLOR= 3: GOSUB 600· IF SC >
 HS THEN HS = SC: GOSUB 1700
330  IF AX <  > 28 THEN  GOSUB
4110
340  RETURN
```

Esta e a rotina que anima a flecha. Ela apaga a imagem anterior e coloca a nova na posição seguinte, determinada



pela variável **AX**. Essa variável é decrementada na linha 300. Para evitar que a flecha seja posta fora da tela, **AX** é recolocada em 29 (28, no Apple e no TK-2000) sempre que atinge um valor inferior a zero. Quando o valor de **AX** é 29 (ou 28), a flecha está com Freddy, e, como o balão não pode ser estourado, a rotina é abandonada.

Logo depois do disparo da flecha — **AX**< >29 —, a linha 310 verifica se ela atingiu o balão e, em caso positivo, aumenta o placar. A rotina da linha 600, que promove o estouro do balão, é, então, chamada.

A linha 330, por sua vez, chama a sub-rotina que desenha a flecha na posição de Freddy.

FREDDY NA ESCADA

S

```
400 LET a$=INKEY$: IF a$=""
THEN  RETURN
410 IF a$="Z" OR a$="z" THEN
GOTO 450
420 IF a$="c" OR a$="C" THEN
GOTO 440
430 IF a$<>" " THEN  RETURN
432 IF ax<>29 THEN  RETURN
434 LET ax=28: PRINT AT ay,29;
" ": RETURN
440 IF my=19 THEN  RETURN
```

```
445 PRINT AT my,30; INK 6;"kl"
· LET my=my+1: PRINT AT ay,29;
" "· IF ax=29 THEN  LET ay=ay+
 1
446 GOTO 470
450 IF my=5 THEN  RETURN
460 PRINT AT my+2,30; INK 6;"
kl"· LET my=my-1: PRINT AT ay,
29;" "  IF ax=29 THEN  LET ay=
ay-1
470 GOSUB 4000: RETURN
```



```
400 IF PEEK(337)=255 THEN RETUR
N
410 IF PEEK(341)=247 THEN 450
420 IF PEEK(342)=247 THEN 440
430 IF PEEK(345)<>247 THEN RETU
RN
432 IF AX<>29 THEN RETURN
434 AX=28:PUT(232,AY*8)-(239,AY
*8+7),S1,PSET:RETURN
440 IF MY=19 THEN RETURN
445 PUT(240,MY*8)-(255,MY*8+7),
KL,PSET:MY=MY+1:PUT(232,AY*8)-(
239,AY*8+7),S1,PSET:IF AX=29 TH
EN AY=AY+1
446 GOTO 470
450 IF MY=5 THEN RETURN
460 PUT(240,MY*8+16)-(255,MY*8+
23),KL,PSET:MY=MY-1:PUT(232,AY*
8)-(239,AY*8+7),S1,PSET:IF AX=2
9 THEN AY=AY-1
470 GOSUB 4000:RETURN
```



```
400 A$=INKEY$
410 IF A$=CHR$(30) THEN 450
420 IF A$=CHR$(31) THEN 440
430 IF A$<>CHR$(32) THEN RETURN
432 IF AX<>29 THEN RETURN
434 AX=28:RETURN
440 IF MY=21 THEN RETURN
445 MY=MY+1:IF AX=29 THEN AY=AY
+1
446 GOTO 470
450 IF MY=5 THEN RETURN
460 MY=MY-1:IF AX=29 THEN AY=AY
-1
470 GOSUB 4000:RETURN
```



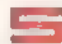
```
400  POKE  - 16368,0: FOR X = 1
 TO 10: IF  PEEK ( - 16384) > 1
27 THEN X = 10: NEXT :PK =  PEE
K ( - 16384): GOTO 410
405  POKE  - 16368,0: NEXT : RE
TURN
410  IF PK = 218 THEN 440
420  IF PK = 193 THEN 450
430  IF PK <  > 160 THEN  RETUR
N
432  IF AX <  > 28 THEN  RETURN
434 AX = 27: HCOLOR= 0: DRAW FL
 AT 224,AY * 8: HCOLOR= 3: RETU
RN
440  IF MY = 19 THEN  RETURN
445  HCOLOR= 0: DRAW FD AT 240,
MY * 8:MY = MY + 1: IF AX = 28
THEN  DRAW FL AT AX * 8,AY * 8:
AY = AY + 1
446  HCOLOR= 3: DRAW FD AT 240,
MY * 8
448  GOTO 470
450  IF MY = 5 THEN  RETURN
460  HCOLOR= 0: DRAW FD AT 240,
MY * 8:MY = MY - 1: IF AX = 28
THEN  DRAW FL AT AX * 8,AY * 8:
AY = AY - 1
465  DRAW FD AT 240,MY * 8
470  HCOLOR= 3: GOSUB 4000: RET
URN
```

Em todos os programas, as linhas 400 a 420 lêem o teclado, enquanto as linhas 430 e 440 verificam se a barra de espaços foi pressionada e se Freddy está com a flecha.

Enquanto nosso personagem se movimenta para cima ou para baixo na escada, os caracteres desta têm que ser repostos (ou a escada acabará desaparecendo). Se **AX** = 29 (28, na versão para o Apple e o TK-2000), a flecha também deve ser movimentada.

Para que o programa funcione adequadamente no TK-2000, os usuários precisarão adaptar o trecho que vai da linha 400 até a linha 405. Consulte o artigo de *Programação BASIC* que trata da substituição da função **INKEY$** nesse microcomputador e faça, então, as alterações que forem necessárias.

AS PERNAS DA ARANHA



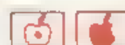
```
500 LET temp=s(xpos)+s(xinc)
510 IF temp<1 OR temp>8+(3-pro
ps)*9 THEN  LET s(xinc)=-s(xin
c): GOTO 500
520 POKE 23607,60· PRINT AT s(
ypos),s(xpos);"  ";AT s(ypos)+
1,s(xpos);"  ": POKE 23607,252
530 LET s(ypos)=s(ypos)+s(yinc
)  LET s(xpos)=temp: LET s(pic
ture)=1-s(picture): GOSUB 4200
540 RETURN
```



```
500 TE=S(XP)+S(XI)
510 IF TE<1 OR TE>8+(3-PP)*9 TH
EN S(XI)=-S(XI):GOTO 500
520 X2=S(XP)*8:Y2=S(YP)*8:PUT(X
2,Y2)-(X2+15,Y2+15),SP,PSET
530 S(YP)=S(YP)+S(YI):S(XP)=TE:
S(PI)=1-S(PI):GOSUB 4200
540 RETURN
```



```
500 TE=S(XP)+S(XI)
510 IF TE<1 OR TE>8+(3-PP)*9 TH
EN S(XI)=-S(XI):GOTO 500
530 S(YP)=S(YP)+S(YI):S(XP)=TE:
S(PI)=1-S(PI):GOSUB 4200
540 RETURN
```

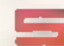


```
500 TE = S(XP) + S(XI)
510  IF TE < 1 OR TE > 8 + (3 -
 PP) * 9 THEN S(XI) =  - S(XI):
 GOTO 500
520 X2 = S(XP) * 8:Y2 = S(YP) *
 8: HCOLOR= 0: DRAW S(PI) + 1 A
T X2,Y2: HCOLOR= 3
530 S(YP) = S(YP) + S(YI):S(XP)
 = TE:S(PI) = 1 - S(PI): GOSUB
4200
540  RETURN
```

A última rotina de animação trata da aranha marciana. Para tornar o jogo mais emocionante, ela não fica parada esperando por seu "almoço". Ao contrário, movimenta-se impacientemente entre a parede e as portas. Desenhamos, por isso, duas figuras para a aranha. O número da figura corrente, manipulado pela linha 530, é guardado em **s (picture)**, no programa do Spectrum, e em **S (PI)**, no dos demais micros.

As linhas 500 e 510 têm a função de impedir que a aranha escape de sua gaiola antes que todas as portas tenham sido removidas.

ERA UMA VEZ UM BALÃO



```
600 PRINT AT b(ypos),b(xpos);
BRIGHT 1; INK b(colour);"gh";
AT b(ypos)+1,b(xpos);"ij"
610 POKE 23607,60
620 PRINT AT 0,14; INK 0;
PAPER 6;score
630 SOUND .5,-20
635 LET bl=bl-1· PRINT AT 0,7;
INK 0; PAPER 6;bl;: IF bl=9
THEN  PRINT  INK 0; PAPER 6;"
"
637 IF bl=0 THEN  LET bl=15+5*
level: LET level=level+1: LET
props=props-1: PRINT  INK 0;
PAPER 6;AT 0,7;bl,AT 0,2;level
: GOSUB 6000
640 PRINT AT b(ypos),b(xpos);"
  ";AT b(ypos)+1,b(xpos);"  "
650 PRINT AT ay,ax;"  ": LET a
x=29: LET ay=my+1
660 POKE 23607,252: GOSUB 4000
: GOSUB 5000: RETURN
```



```
600 X2=B(XP)*8:Y2=B(YP)*8:PUT (
X2,Y2)-(X2+15,Y2+15),GJ,PSET
620 COLOR 0:LINE(114,2)-(150,7)
,PSET,BF:NU=SC:DRAW"C1;BM114,2"
:GOSUB 1650
630 PLAY"V31;T200;O2;BAGFEDC;O1
;BAGFEDC"
635 BL=BL-1:COLOR0:LINE(58,2)-(
68,7),PSET,BF:DRAW"BM58,2;S2;C1
":NU=BL:GOSUB 1650
637 IF BL=0 THEN PP=3:LV=LV+1:B
```

```
L=15+5*LV:COLOR0:LINE(14,2)-(24
,7),PSET,BF:LINE(58,2)-(68,7),P
SET,BF:NU=LV:DRAW"BM14,2;C1":GO
SUB 1650:GOSUB 6000:NU=BL:DRAW"
BM58,2;C1" GOSUB 1650
640 X2=B(XP)*8:Y2=B(YP)*8:PUT(X
2,Y2)-(X2+15,Y2+15),SP,PSET
650 PUT(AX*8,AY*8)-(AX*8+15,AY*
8+7),S2,PSET:AX=29:AY=MY+1
660 GOSUB 4000:GOSUB 5000:RETUR
N
```



```
600 X2=B(XP)*8·Y2=B(YP)*8 PUT S
PRITE 4,(X2,Y2),9,BE
630 BL=BL-1
640 IF BL=0 THEN PP=3 LV=LV+1·B
L=.5+5*LV
650 GOSUB 1700
660 PUT SPRITE 3,(AX*8,AY*8),4,
FL·AX=29·AY=MY
670 GOSUB 4000·GOSUB 5000·RETUR
N
```



```
600  HCOLOR= 3 X2 = B(XP) * 8·Y
2 = B(YP) * 8· DRAW BE AT X2,Y2

610 BL = BL · 1
620  IF BL = 0 THEN PP = 3·LV
 LV + 1·BL = 15 + 5 * LV
630  GOSUB 1700
640  HCOLOR= 0·X2 = B(XP) * 8 Y
2 = B(YP) * 8: DRAW BE AT X2,Y2

650  DRAW FL AT AX * 8,AY * 8·A
X = 28:AY = MY + 1: HCOLOR= 3
660  GOSUB 4000. GOSUB 5000. RE
TURN
```

Essa rotina promove o estouro do balão quando ele é atingido pela flecha. Ela é muito simples: apenas coloca na tela a imagem do balão estourado, apagando-a em seguida.

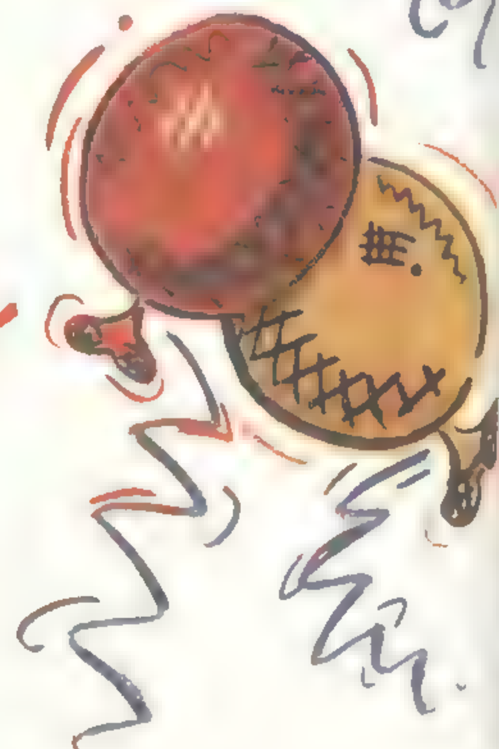
Depois de cada estouro, atualiza-se o número de balões restantes e, se for necessário, também o nível de dificuldade. A flecha deve voltar, então, à posição de Freddy, para que ele possa estourar o próximo balão.











ALAVANCAS, POLIAS E ROLDANAS

- A ALAVANCA
- DISTÂNCIA X PESO
- SISTEMAS DE POLIAS
- POUPANDO ESFORÇOS
- O ELEVADOR HIDRÁULICO

A mecânica é a área da física que estuda as forças que interagem entre os corpos. Utilize o computador para analisar alguns de seus princípios, simulando situações da vida cotidiana.

Os grandes monumentos antigos — como as pirâmides do Egito e os templos incas — sempre nos intrigaram, pois os povos que os construíram não possuíam a avançada tecnologia de que dispomos nos dias de hoje.

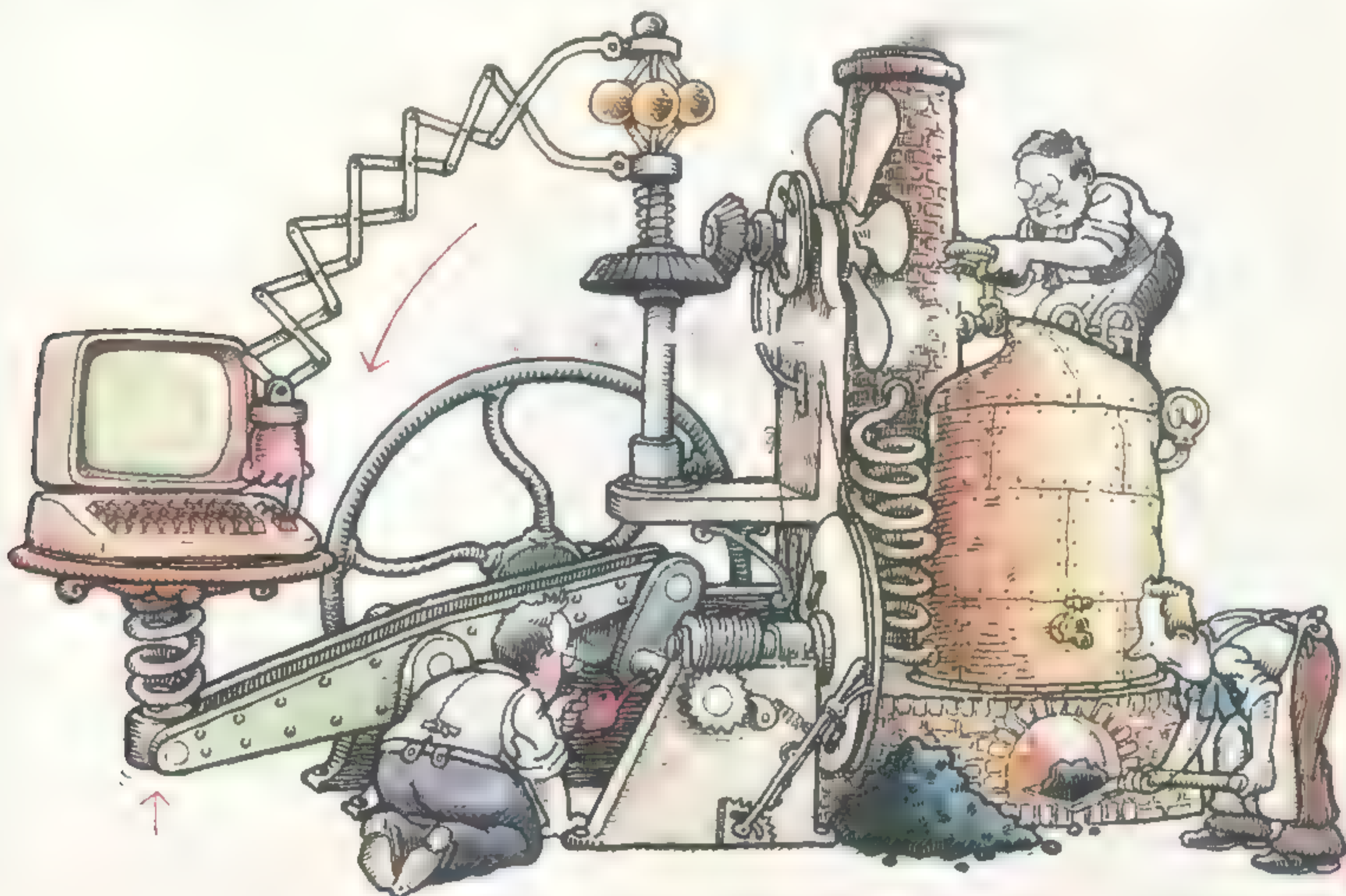
Nenhum construtor contemporâneo se atreveria a executar uma obra como aquelas sem ter em mãos um equipamento que o ajudasse a cortar, transportar e erguer a grandes alturas as gigantescas pedras que as compõem. No entanto, nossas mais sofisticadas máquinas são projetadas a partir dos mesmos princípios que orientavam os antigos engenheiros. A ciência que estuda tais princípios é a mecânica.

As forças que atuam sobre um sistema estático (como um edifício) ou dinâmico (como as engrenagens de uma máquina) são percebidas de modo quase instintivo. Assim, quando levantamos uma xícara de café, calculamos automaticamente a força exata para fazê-lo sem derramar o líquido. Do mesmo modo, ao escolhermos uma tábua para uma estante, sabemos mais ou menos qual a espessura que ela deve ter para suportar um determinado peso.

Em situações simples, como as mencionadas, apenas estimamos as forças em questão. Porém, em alguns casos, os cálculos devem ser muito precisos. É aí que entram os computadores.

Você pode, é claro, usar dados obtidos através de suas próprias experiências para planejar simulações envolvendo forças. Assim, em programas de jogos, é possível estimar, com uma boa margem de acerto, a que distância uma bola vai parar depois de ter sido golpeada por um bastão.

Mas o computador em geral requer dados muito precisos. Quando um engenheiro está projetando uma ponte, por exemplo, ele precisa saber exatamente a força que atua sobre cada estrutura e o seu limite máximo de resis-



tência. Aprendemos, aqui, a calcular forças em sistemas simples, onde as cargas podem ser variáveis.

A ALAVANCA

O transporte de cargas muito pesadas torna-se um sério problema à medida que vão sendo construídas estruturas cada vez maiores. Durante a década de 60, o foguete Saturno V, que levaria o homem até a Lua, era transportado em uma grande plataforma móvel, a maior até então fabricada.

Atualmente, as plataformas de petróleo do mar do Norte são transportadas por terra em veículos que flutuam sobre um colchão de ar. Esses equipamentos, e muitos outros mais simples, dependem sempre de algum tipo de máquina que lhes dê pressão, tração ou movimento. Essas máquinas, por sua vez, não passam de montagens mais complexas de alguns instrumentos básicos, já conhecidos e usados há séculos.

Entre os mais primitivos deles está a alavanca. Em sua forma mais simples, ela consiste de um bastão. Uma das extremidades desse bastão fica embaixo da carga e a outra é levantada para empurrá-la. Usando esse artifício, uma pessoa pode mover objetos muitas vezes mais pesados que ela. Como afirmava Arquimedes, o grande matemático do mundo antigo, poderíamos levantar a Terra, se tivéssemos um bastão bem longo e um ponto de apoio.

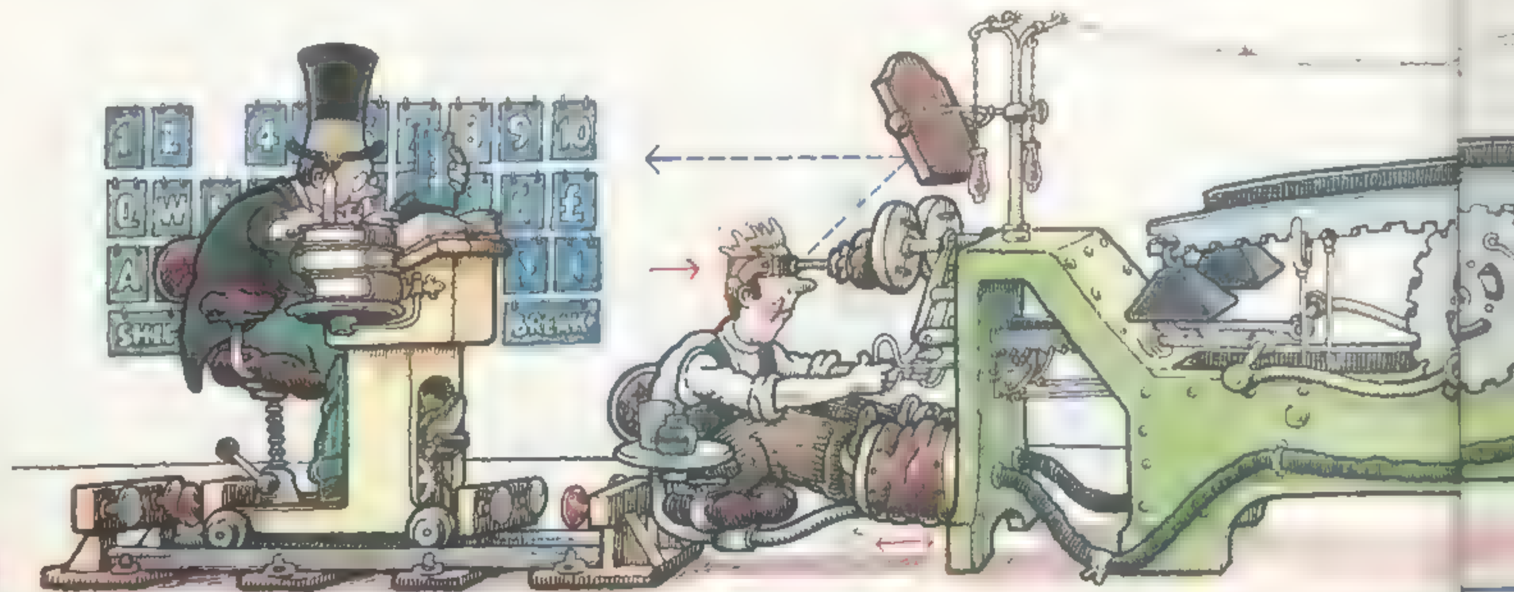
Esse ponto de apoio está situado em algum lugar entre as duas extremidades do bastão. Digite este programa para ver uma simulação do funcionamento de uma alavanca.



```
30 BORDER 0: PAPER 0: INK 7:
CLS : OVER 1
40 INPUT "Distancia do apoio
a partir da  esquerda    (1-8
m) ";d
50 IF d<1 OR d>8 THEN  GOTO
40
60 LET w=(10-d)/d
70 PRINT AT 1,0;"Peso necessa
rio para equilibrar 100 KG=";
w*100;"KG"
100 PLOT  INK 4,0,15: DRAW
INK 4;255,0: FOR n=0 TO 55:
PLOT (28+20*d)-n/3,71-n
110 DRAW  INK 7;2*((28+20*d)-
PEEK 23677),0: NEXT n
120 LET a=d-10: LET a=ATN (a/(
1-a*a))+2*ATN (1)
130 FOR b=a TO 2*ATN (1) STEP
(2*ATN (1)-a)/10
140 FOR k=1 TO 2: GOSUB 1000:
GOSUB 1500
160 NEXT k
170 NEXT b
180 LET B=2*ATN (1): GOSUB
1000: GOSUB 1500
190 IF INKEY$="" THEN  GOTO
190
200 RUN
1000 PLOT 120-100*SIN (b),71-20
*d*COS (b)
1011 DRAW 127+100*SIN (b)-PEEK
23677,71+(200-20*d)*COS (b)-PEE
K 23678
1025 DRAW 0,-8: DRAW -10,0: DRA
W 0,-10: DRAW 20,0: DRAW 0,10:
DRAW -9,0: POKE 23678,(PEEK 236
78)+6
1030 RETURN
1500 PLOT 128-100*SIN (b),70-20
*d*COS (b)
1510 LET e=SQR (SQR w)
1520 DRAW 0,-8: DRAW -10*e,0: D
RAW 0,-10*e: DRAW 20*e,0: DRAW
0,10*e: DRAW -9*e,0
1530 RETURN
```



```
10 PMODE 3,1 PCLS
20 COLOR 3:LINE (0,180)-(255,19
1),PSET,BF
30 CLS
40 INPUT" DISTANCIA DO APOIO A
PARTIR DA  ESQUERDA (1 8 M)  ";
D
50 IF D<1 OR D>8 THEN 30
60 W=(10-D)/D
70 PRINT PRINT"PESO NECESSARIO
PARA EQUILIBRAR 100 KG =";W*100
,"KG"
80 FOR G=1 TO 4000:NEXT
90 SCREEN 1,0
100 COLOR 2 LINE(28+20*D,150)-(
18+20*D,179),PSET
110 LINE  (38+20*D,179),PSET:LI
NE -(28+20*D,150),PSET:PAINT(28
+20*D,160),2
120 A=10-D A ATN(A*A-1)
130 FOR AN=A TO 2*ATN(1) STEP (
2*ATN(1) A)/10
140 C=4 GOSUB 1000 C 3 GOSUB 15
0
150 FOR G=1 TO 500:NEXT
160 C 1:GOSUB 1000 GOSUB 1500
170 NEXT
180 AN 2*ATN(1) C=4 GOSUB 1000
  3 GOSUB 1500
190 IF INKEY$="" THEN 190
200 RUN
1000 COLOR C LINE(128 100*SIN(A
N),149 20*D*COS(AN))-(134+100*S
IN(AN),149+(200 20*D)*COS(AN)),
PSET
1020 DRAW"D2L6D6R10U6L8"
1030 RETURN
1500 COLOR 3 LINE(132-100*SIN(A
N),147-20*D*COS(AN))-(132-100*S
IN(AN) SQR(W)*7,147-20*D*COS(AN
)-SQR(W)*7),PSET,BF
1530 RETURN
```

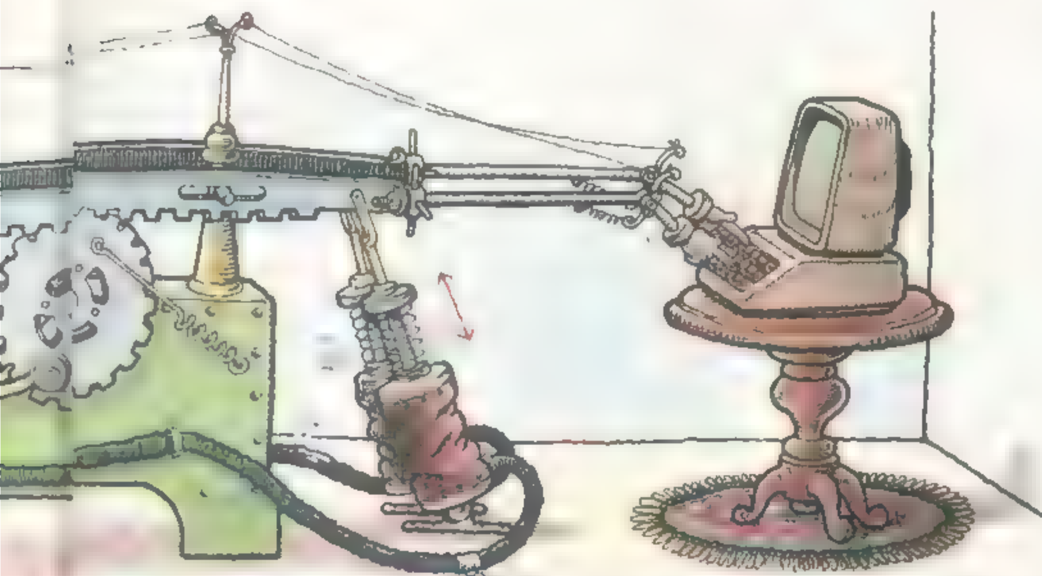




```
5 SCREEN 0:MAXFILES=1:COLOR 15,
1,1
10 INPUT"distância do ponto de
apoio (1-8m) ";D
15 IF D<1 OR D>8 THEN 10
20 W=(10-D)/D
25 SCREEN 2·OPEN"GRP:" FOR OUTP
UT AS #1
30 PRESET(0,0)
35 PRINT #1,"PESO QUE EQUILIBRA
 100 KG=";INT(W*100);"KG"
40 LINE(0,180)-(255,191),3,BF
45 FOR I=0 TO 10
50 LINE (28+20*I,180)-(28+20*I,
191),6
55 NEXT I
60 LINE(28+20*D,150)-(18+20*D,1
79),2·LINE-(38+20*D,179),2:LINE
-(28+20*D,150),2·PAINT(28+20*D,
160),2
65 A=10-D·A=ATN(A*A-1)
70 FOR AN=A TO 2*ATN(1) STEP (2
*ATN(1)-A)/10
80 C=4 GOSUB 1000 C=3·GOSUB 1500
90 FOR G=1 TO 500·NEXT G
100 C=1·GOSUB 1000·GOSUB 1500
110 NEXT AN
120 AN=2*ATN(1)·C=4·GOSUB 1000·
C=3 GOSUB 1500
130 IF INKEY$="" THEN 130
140 RUN
1000 LINE (128 100*SIN(AN),149-
20*D*COS(AN))-(134+100*SIN(AN),
149+(200-20*D)*COS(AN)),C
1010 DRAW"D2L6D6R10U6L8"
1020 RETURN
1500 LINE (132-100*SIN(AN),147-
20*D*COS(AN))-(132-100*SIN(AN)-
SQR(W)*7,147-20*D*COS(AN)-SQR(W
)*7),C,BF
1510 RETURN
```



```
5  HOME : HGR : HCOLOR= 3
10  VTAB (23): INPUT "DISTANCIA
 DO PONTO DE APOIO(1-8M)=";D
15  IF D < 1 OR D > 8 THEN 5
20 W = (10 - D) / 10
30  VTAB (24): PRINT "PARA EQUI
LIBRAR 100 KG E PRECISO ";W * 1
00;" KG"
40  HPLOT 28 + 20 * D,120 TO 18
 + 20 * D,149
50  HPLOT  TO 38 + 20 * D,149 T
O 28 + 20 * D,120
60 A = 10 - D:A =  ATN (A * A -
 1)
70  FOR AN = A TO 2 *  ATN (1)
STEP (2 *  ATN (1) - A) / 10
80  HCOLOR= 3: GOSUB 1000
90  FOR G = 1 TO 1000: NEXT G
100  HCOLOR= 0· GOSUB 1000
110  NEXT AN
120 AN = 2 *  ATN (1)· HCOLOR=
3: GOSUB 1000
130  GET A$: RUN
1000 X = 128 - 100 *  SIN (AN):
Y = 119 - 20 * D *  COS (AN):R
= 134 + 100 *  SIN (AN):S = 119
 + (200 - 20 * D) *  COS (AN)
1010  HPLOT X,Y TO R,S
1020  HPLOT R,S TO R,S + 2 TO R
 - 6,S + 2 TO R - 6,S + 8 TO R
+ 6,S + 8 TO R + 6,S + 2 TO R,S
 + 2
1500  HPLOT 132 - 100 *  SIN (A
N),117 - 20 * D *  COS (AN) TO
132 - 100 *  SIN (AN),117 - 20
* D *  COS (AN) -  SQR (W) * 7
TO 132 - 100 *  SIN (AN) -  SQR
 (W) * 7,117 - 20 * D *  COS (A
N) -  SQR (W) * 7
1510  HPLOT 132 - 100 *  SIN (A
N) -  SQR (W) * 7,117 - 20 * D
*  COS (AN) -  SQR (W) * 7 TO 1
32 - 100 *  SIN (AN) -  SQR (W)
 * 7,117 - 20 * D *  COS (AN) T
O 132 - 100 *  SIN (AN),117 - 2
0 * D *  COS (AN)
1520  RETURN
```



Ao executar o programa, o computador pedirá que você digite a distância do ponto de apoio (**D**), a partir da extremidade esquerda de uma barra de 10 m. Ele imprime, então, o peso necessário para equilibrar 100 kg na outra ponta (**W**). O ponto de apoio é desenhado pelas linhas 100 e 110 (TRS-Color e Spectrum), 40 e 50 (Apple e TK-2000) e 60 (MSX). Um laço controla a animação da alavanca — linhas 130 a 170 (TRS-Color e Spectrum) e 70 a 110 (MSX, Apple e TK-2000) —, chamando a sub-rotina 1000, que desenha a barra, e a 1500, que desenha os pesos.

Introduza diferentes valores para **D** e note que, quanto mais distante do ponto de apoio, menor é o peso necessário. Do mesmo modo, uma distância mais curta requer um peso maior.

DISTÂNCIA X PESO

Existe uma fórmula matemática bem simples para o cálculo de distâncias e pesos em alavancas; basta que você saiba onde se encontra o ponto de apoio. Segundo essa fórmula, a carga, multiplicada pela distância até o ponto de apoio, é igual ao peso usado vezes a distância até esse ponto. Em termos de variáveis, seria o seguinte:

$$C \times DC = P \times DP$$

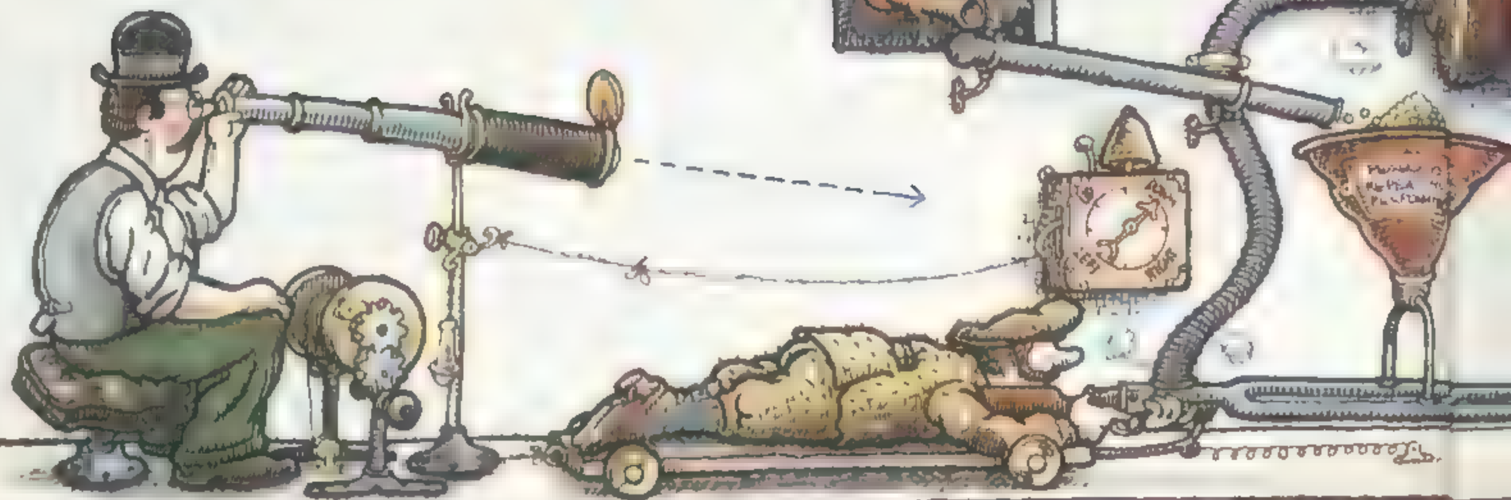
No programa anterior, usamos **W** no lugar de **P**. A barra mede 10 m, logo, **DC** é igual a 10 menos **DP**. Se a carga fosse de 1 kg, o peso necessário para equilibrá-la seria então:

$$W = (10 - DE)/DE$$

conforme usamos nas linhas 60 (TRS-Color e Spectrum) e 20 (MSX, Apple e TK-2000). A variável **DE** (**D**, no programa) foi introduzida por você.

As balanças mecânicas utilizam esse princípio. Toma-se um ponto de apoio no meio da barra e coloca-se o peso a ser medido em uma das extremidades. A partir daí, adicionam-se massas já conhecidas do outro lado, até atingir o equilíbrio. O peso do objeto será a soma dessas massas. Isso funciona muito bem para pequenos pesos, mas, quando se trata de grandes cargas, o procedimento é outro. Nesse caso, o ponto de apoio é colocado bem próximo da carga e calcula-se o peso aplicando a fórmula já explicada.

Depois de ter brincado um pouco com seu programa, você estará pronto para identificar o funcionamento de alavancas em qualquer tipo de máquina.



Ao trabalhar com uma chave inglesa ou com o macaco do seu carro, nunca se esqueça de que a distância que você tomar do ponto de apoio poderá reduzir muito o seu esforço.

SISTEMAS DE POLIAS

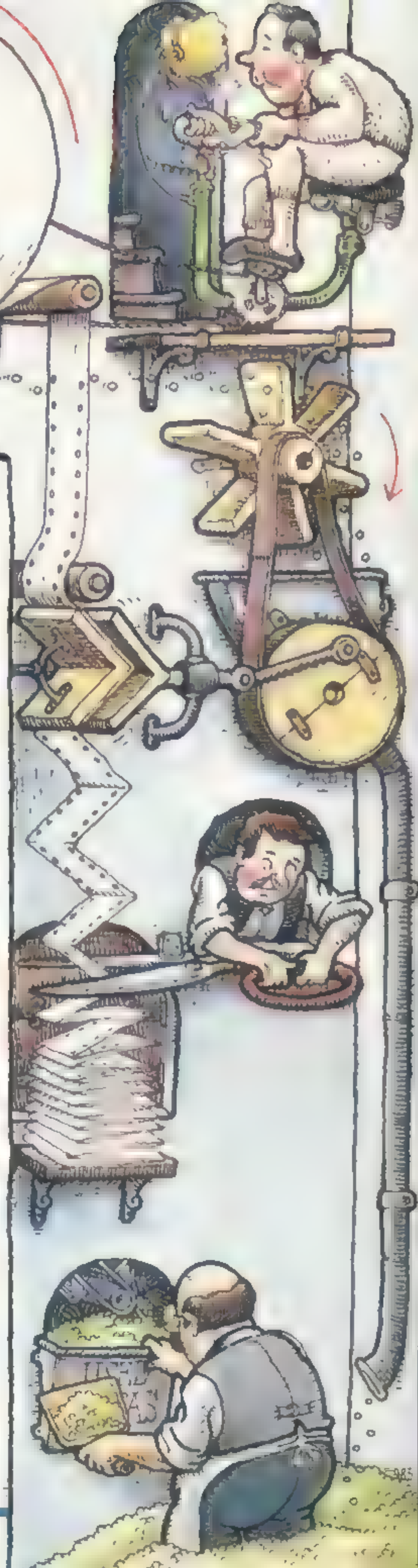
As alavancas são muito úteis para puxar e empurrar objetos, mas, quando se quer levar uma carga a grandes alturas, é necessário utilizar as polias. Digite o programa a seguir para vê-las em funcionamento.

S

```
10 CLEAR 32399: RESTORE :
GOSUB 510· BORDER 0. PAPER 0:
 INK 7: CLS
20 PRINT AT 10,0;"Quantas rol
danas (2, 4 ou 6)?"
30 LET a$=INKEY$: IF a$<>"2"
AND a$<>"4" AND a$<>"6" THEN
GOTO 30
40 PRINT TAB (10);a$: LET np=
VAL (a$)· LET l=25*np-50
45 IF np=2 THEN  POKE 32431,
25
46 IF np=4 THEN  POKE 32431,
17
47 IF np=6 THEN  POKE 32431,
14
50 PRINT · PRINT "Sao necessa
rios ";INT (1000/np);" quilog
ramas" PRINT "para levantar
1 tonelada"
55 FOR m=1 TO 500· NEXT m
60 CLS : GOSUB 1000
70 LET sp=120
90 FOR k=1 TO 50
100 RAND USR 32400
105 PLOT  OVER 1;191-l-np,sp·
DRAW  OVER 1,8,0
110 PLOT  OVER 1;191-l-np,sp·
DRAW  OVER 1;8,0
120 LET sp=sp-np· IF sp<=0
THEN  LET sp=120
150 NEXT k
160 IF INKEY$="" THEN  GOTO
160
170 RUN
510 FOR n=32400 TO 32491
520 READ a: POKE n,a
530 NEXT n
540 DATA 62,60,79,230,192,15,
15,15,198,64,103,121,230,7,132
,103,121,135,135,230,224,111,
62
550 DATA 175,254,192,208,145,
216,8,14,14,125,177,111,62,30,
254,32,208,145,216,60,79,6,0,
197,229,17,224,91,237,176,225
560 DATA 193,217,8,167,40,30,
71,217,124,60,87,93,230,7,32,
10,123,198,32,95,56,4,122,214,
8,87,235
570 DATA 229,197,237,176,193,
225,217,16,227,217,201
580 RETURN
1000 PLOT 0,170: DRAW 255,0
1010 FOR k=1 TO np STEP 2
1020 CIRCLE (232-k*26),140,13
1030 CIRCLE (258-k*26),50,13
1040 NEXT k
1050 PLOT 245,170· DRAW 0,-125
1060 FOR k=1 TO np-1
1070 PLOT 246-k*26,140· DRAW 0,
-90
1080 NEXT k
1090 PLOT 197-l-np,140: DRAW 0,
 140
1100 PLOT 208,141: DRAW (256-np
*26)-PEEK 23677,0
1110 PLOT 234,50: DRAW (282-np*
26)-PEEK 23677,0
1120 PLOT 206,170: DRAW 0,-28:
PLOT 258-np*26,170: DRAW 0,-28
1130 PLOT 231-l*2/3,50: DRAW 0,
-20
1140 PLOT 232-l/3,50: DRAW 0,-2
0
1145 DRAW (231-l*2/3)-PEEK 2367
7,0
1150 PLOT 231-l/2,29: DRAW 0,-1
0
1160 DRAW -9,0: DRAW 0,-10: DRA
W 19,0: DRAW 0,10: DRAW -9,0
1170 PLOT 180-l,0: DRAW (180-l-
20/SQR (np))-PEEK 23677,0· DRAW
 0,(20/SQR (np))-PEEK 23678: DR
AW (180-l) PEEK 23677,0· DRAW 0
,0-PEEK 23678
1210 RETURN
```



```
10 PMODE 3,1:DIM P(280)
20 CLS:PRINT "QUANTAS ROLDANAS
(2, 4 OU 6) ? ";
30 A$=INKEY$:IF A$<>"2" AND A$<
>"4" AND A$<>"6" THEN 30
40 PRINT A$:NP=VAL(A$):L=25*NP-
50
50 PRINT·PRINT"SAO NECESSARIOS
";1000/NP:PRINT"QUILOGRAMAS PAR
A LEVANTAR 1 TON.":FOR G=1 TO 4
000·NEXT
60 PCLS·SCREEN 1,0·GOSUB 1000
70 GET(249,172)-(215-L,106),P,G
 SP=38
80 COLOR 4,1
90 FOR K=106 TO 49 STEP -1
100 PUT (249,K+66)-(215-L,K),P,
PSET
110 LINE(191-L-NP,SP)-(199-L-NP
,SP),PRESET
120 SP=SP+NP·IF SP>191 THEN SP=
38
130 LINE(195-L-NP,33)-(195-L-NP
,191),PSET
140 LINE(191-L-NP,SP)-(199-L-NP
,SP),PSET
150 NEXT
160 IF INKEY$="" THEN 160
170 RUN
1000 LINE(0,0)-(255,10),PSET,BF
1010 FOR K=1 TO NP STEP 2
1020 CIRCLE(232-K*26,33),13,2·P
AINT(232-K*26,33),2
1030 CIRCLE (232-K*26,33),14,4,
1,.5,1
1040 CIRCLE (258-K*26,121),13,2
·PAINT (258-K*26,121),2
1050 CIRCLE (258-K*26,121),14,4
,1,0,.5
1060 NEXT
1070 LINE(245,10)-(245,121),PSE
T
1080 FOR K=1 TO NP-1
1090 LINE (246-K*26,33)-(246-K*
26,121),PSET
1100 NEXT
```



```
1110 LINE(195-L-NP,33)-(195 L N
P,191),PSET
1120 COLOR 4,3.LINE(208,32)-(25
6-NP*26,34),PRESET,BF
1130 LINE(234,120) (282-NP*26,1
22),PRESET,BF
1140 LINE(206,10)-(206,33),PSET
:LINE(258-NP*26,10)-(258-NP*26,
33),PSET
1150 LINE(232-L*2/3,121)-(232-L
*2/3,141),PRESET
1160 LINE(232-L/3,121)-(232-L/3
,141),PRESET
1170 LINE -(232-L*2/3,141),PRES
ET
1180 LINE(232 L/2,141)-(232-L/2
,151),PRESET
1190 DRAW"L9D19R19U19L9"
1200 LINE(180 L,191) (180-L-20/
SQR(NP),191 20/SQR(NP)),PRESET,
BF
1210 RETURN
```



```
100  SCREEN 0:COLOR 15,1,1:PRIN
T TAB(7)"Quantas polias(2,4 ou
6)?";
110 A$=INKEY$:IF A$<>"2" AND A$
<>"4" AND A$ <>"6" THEN 110
120 PRINT A$:NP=VAL(A$):L=25*NP
-50
130 PRINT: PRINT TAB(7)"Força n
ecessária =";INT(1000/NP);"Kg":
PRINT TAB(7)"para levantar 1 to
nelada":FOR G=1 TO 4000:NEXT
1000 CLS:SCREEN 2
1010 LINE(0,0)-(255,10),5,BF
1020 FOR K=1 TO NP STEP 2
1030 CIRCLE(232-K*26,33),13,2:P
AINT(232-K*26,33),2
1040 LINE(232-K*26,33)-(232-K*2
6,10)
1050 NEXT K
1060 LINE(208,32)-(256-NP*26,34
),,BF
1070 FOR V=0 TO -30 STEP -5
1080 GOSUB 1500
1090 FOR I=1 TO 300:NEXT I
1100 NEXT V
1110 FOR I=1 TO 2000:NEXT I
1120 GOTO 100
1500 LINE(194-L-NP-12/NP,75-3*(
V+5))-(196-L-NP+12/NP,75+30/NP-
3*(V+5)),1,BF
1510 LINE(195-L-NP,33)-(195-L-N
P,75-3*V),2
1520 LINE(194 L-NP-12/NP,75-3*V
)-(196-L-NP+12/NP,75+30/NP-3*V)
,10,BF
1530 LINE(249,175+V)-(215-L,107
+V),1,BF
1540 FOR K=1 TO NP STEP 2
1550 CIRCLE(258-K*26,121+V),13,
2:PAINT(258-K*26,121+V),2
1560 NEXT K
1570 LINE(245,10+V)-(245,121+V)
,2
1580 FOR K=1 TO NP-1
1590 LINE(246-K*26,35)-(246-K*2
6,121+V),2
1600 NEXT K
1610 LINE(234,120+V)-(282-NP*26
,122+V),,BF
1620 LINE(232-L*2/3,121+V)-(232
-L*2/3,141+V)
1630 LINE(232-L/3,121+V)-(232-L
/3,141+V)
1640 LINE-(232-L*2/3,141+V)
1650 LINE(232-L/2,141+V)-(232-L
/2,151+V)
1660 DRAW"C8L9D19R19U19L9"
1670 RETURN
```



```
10 E = 780
20 F =  INT (E / 256)
30  POKE 232,E - F * 256: POKE
233,F
40  FOR I = E TO E + 41 + 1 * 3
■
50  READ A· POKE I,A
60  NEXT
70  SCALE= 1: ROT= 0
80  DATA   20 ,0 ,42 ,0 ,74 ,0
,106 ,0 ,138 ,0 ,170 ,0 ,202 ,0
 ,234 ,0 ,10 ,1 ,42 ,1 ,74 ,1 ,
106 ,1 ,138 ,1 ,170 ,1 ,202 ,1
,234 ,1 ,10 ,2 ,42 ,2 ,74 ,2 ,1
06 ,2 ,138 ,2
90  DATA    0 ,72 ,45 ,109 ,209
 ,251 ,219 ,23 ,77 ,73 ,169 ,25
1 ,219 ,27 ,110 ,73 ,9 ,213 ,22
```

```
3 ,219 ,115 ,77 ,9 ,141 ,219 ,6
3 ,255 ,2 ,0 ,0 ,0 ,0
100  TEXT : HOME : VTAB (10): HTAB (7): PRINT "QUANTAS POLIAS ? (2,4 OU 6) ";: GET NP
110  IF NP <  > 2 AND NP <  > 4 AND NP <  > 6 THEN 100
120  PRINT NP
130  HGR : HCOLOR= 3: VTAB (23): PRINT "FORCA NECESSARIA=";  INT (1000 / NP)," Kg": PRINT "PARA LEVANTAR 1 TONELADA"
1000  FOR I = 0 TO 9
1010  HPLOT 0,I TO 255,I
1020  NEXT I
1030  FOR K = 1 TO NP STEP 2
1040  DRAW 1 AT 147 - K * 8,33
1060  NEXT K
1070  FOR I = 1 TO NP STEP 2
1080  HPLOT 135 - (I - 2) * 8,10 TO 135 - (I - 2) * 8,34
1090  NEXT I
1100  HPLOT 143,34 TO 143 - (NP - 2) * 8,34
1110  FOR V = 0 TO 36 STEP 3
1115  HCOLOR= 3: GOSUB 1500
1120  FOR T = 1 TO 500: NEXT T
1125  IF V = 36 THEN  GOTO 1150
1130  HCOLOR= 0: GOSUB 1500
1140  NEXT V
1150  FOR T = 1 TO 2000: NEXT T

1160  GOTO 100
1500  FOR K = 1 TO NP STEP 2
1510  DRAW 1 AT 155 - K * 8,121 - V
1520  NEXT K
1530  HPLOT 155,10 TO 155,123 - V
1540  FOR K = 1 TO NP - 1
1550  HPLOT 155 - K * 8,34 TO 155 - K * 8,123 - V
1560  NEXT K
1570  HPLOT 138 - (NP - 2) * 8,34 TO 138 - (NP - 2) * 8,38 + NP / 2 * V
1580  FOR I =  - 1 TO 1
1590  HPLOT 138 - (NP - 2) * 8 + I,38 + NP / 2 * V TO 138 - (NP - 2) * 8 + I,38 + 36 / NP + NP / 2 * V
1600  NEXT I
1610  HPLOT 151,123 - V TO 151 - (NP - 2) * 8,123 - V
1620  HPLOT 151 - (NP / 2 - 1) * 8,123 - V TO 151 - (NP / 2 - 1) * 8,128 - V
1630  FOR I =  - 4 TO 4
1640  HPLOT 151 - (NP / 2 - 1) * 8 + I,128 - V TO 151 - (NP / 2 - 1) * 8 + I,135 - V
1650  NEXT I
1660  RETURN
```

O programa começa perguntando se você quer um sistema de duas, quatro ou seis polias. Em seguida ele mostra na tela o peso necessário para levantar uma carga de 1000 kg. A partir da linha 1000, o computador desenha as roldanas fixas, e, na linha 1500 (na versão para o MSX, Apple e TK-2000) ou 90 (TRS-Color e Spectrum), ele constrói e movimenta as roldanas e os pesos.

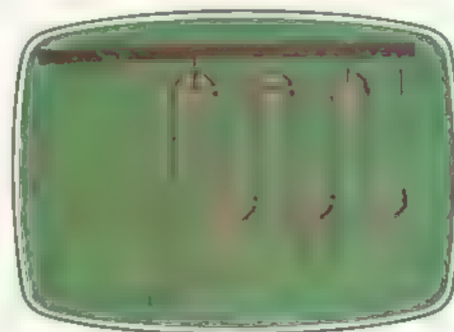
No programa para o Apple e o TK-2000, as linhas 10 a 90 introduzem uma tabela de dados que permite desenhar círculos por meio do comando **DRAW**.

Ao executar o programa, você observará que quanto maior for o número de polias menor será o peso necessário para levantar a carga. Para calcular o peso em um sistema desse tipo, basta dividir a carga pelo número de polias. Se você quiser levantar 1000 kg, usando seis polias, deverá empregar um peso de aproximadamente 166 kg. Em alguns casos ocorre o contrário: sabe-se quanto pesa a carga e qual a força máxima disponível. Nesse caso, divide-se a carga pela força e arredonda-se o resultado para o maior número par. Por exemplo, você sabe que consegue levantar 60 kg usando só uma polia; logo, para levantar 250 kg, você deverá utilizar seis polias.

POUPANDO ESFORÇOS

Como no caso da alavanca, também em um sistema de polias a força está relacionada com a distância. Você pode notar, em nossa simulação, que quanto maior é o número de polias mais a extremidade livre da corda se aproxima do chão. Isso mostra que, para levantar um peso a uma certa altura usando uma força menor, você terá que puxar, para compensar, uma corda de comprimento maior. Na verdade, em um sistema desse tipo, o número de polias já determina qual será o comprimento da corda que precisaremos puxar para que a carga suba em uma unidade. Por exemplo, em um sistema de duas polias, temos que puxar dois metros de corda para que a carga suba um metro.

O sistema que nosso programa simula na tela do microcomputador não é o mais comum. Em geral, as barras passam pelo centro das roldanas, mantendo-as paralelas e bem próximas. Um sistema de seis polias seria visto, então, como três discos colados um ao outro, na vertical, e, abaixo deles, pendurados pelas cordas, mais três discos, onde se prende a carga.



Simulação das polias no MSX.

O ELEVADOR HIDRÁULICO

Outro instrumento que funciona da mesma maneira que a alavanca é o elevador hidráulico. Ele é utilizado para levantar grandes cargas — de automóveis até foguetes e aviões.

O freio dos automóveis atuais, que funciona baseado em um princípio idêntico, mostra como a força de nossos pés pode ser amplificada a ponto de parar, em um tempo muito reduzido, o movimento de um corpo tão pesado.

SIMULAÇÃO

Para observar o funcionamento do elevador hidráulico, digite e rode o programa a seguir.

```
30 BORDER 0: PAPER 7: INK 0: CLS
50 GOSUB 300
90 INPUT "Curso do embolo (1-90) ?  ";tr
100 IF tr<1 OR tr>90 THEN GOTO 90
110 FOR k=1 TO tr
120 PLOT 40,128-(k-1): DRAW INK 7;10,0: PLOT 40,128-k: DRAW 15,0
130 PLOT 175,127+(k-1)/10: DRAW  INK 1;56,0: PLOT 175,127+k/10: DRAW  INK 1;56,0
135 PRINT  INK 0;AT 3,5;k; INK 0;AT 3,25;INT (k/10)
140 NEXT k
150 INK 0: PRINT AT 21,0;"Novamente  ? (s  OR n)       "
160 IF INKEY$="s" THEN  RUN
170 IF INKEY$<>"n" THEN  GOTO 160
180 STOP
300 FOR n=6 TO 18: PRINT PAPER 1;AT n,5;"  ": NEXT n
310 FOR n=6 TO 18: PRINT PAPER 1;AT n,22;"         ": NEXT n
320 FOR n=18 TO 21: PRINT PAPER 1;AT n,5;"                  ": NEXT n
330 PLOT 39,155: DRAW 0,-155: DRAW 192,0: DRAW 0,155: PLOT 56,155: DRAW 0,-124: DRAW 120,0: DRAW 0,124
340 PLOT 40,127: DRAW -2,0: PRINT AT 6,3;"0": PLOT 40,37: DRAW -2,0: PRINT AT 17,2;"90": PLOT 232,127: DRAW 2,0: PRINT
```

```
AT 6,30;"0"
350 PLOT 232,137: DRAW 2,0:
PRINT AT 4,30;"9"
360 FOR n=127 TO 37 STEP -10
370 PLOT 40,n: DRAW -2,0: NEXT
n
380 PLOT 232,132: DRAW 2,0
390 PLOT 120,35: DRAW 0,-35
400 PLOT 110,40· DRAW 20,0:
DRAW -5,5: DRAW 5,-5: DRAW -5,
-5
410 INK 7
440 RETURN
```



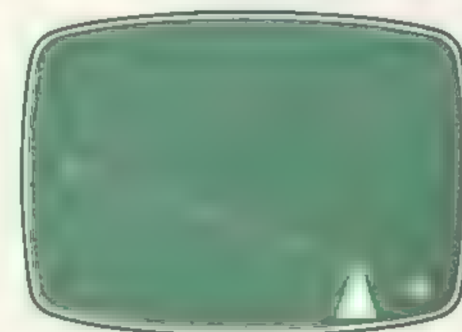
```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 DIM P(4),R(31)
50 GOSUB 300
60 GET(32,52)-(43,38),P,G
70 GET(182,52)-(223,23),R,G
80 IF INKEY$="" THEN 80
90 CLS:INPUT"DESLOCAMENTO DO EM
BOLO    (1-90)";TR
100 IF TR<1 OR TR>90 THEN 90
110 SCREEN 1,0:FOR K=1 TO TR
120 PUT(32,52+K)-(43,38+K),P,PS
ET
130 PUT(182,52-K/10)-(223,23-K/
10),R,PSET
140 NEXT
150 IF INKEY$="" THEN 150
160 CLS:PRINT " NOVAMENTE (S/N)
 ?"
170 A$=INKEY$·IF A$<>"S" AND A$
<>"N" THEN 170
180 IF A$="S" THEN RUN
190 CLS:END
300 DRAW"BM30,30C2D131R195U131B
L45D20NR45D96L135U96NL15U20"
310 PAINT(200,70),3,2
320 DRAW"BM32,50C3R10BR140R41"
330 DRAW"BM127,161C2U20BH8C4R16
NH5G5"
340 LINE(34,48)-(41,41),PSET,BF
350 LINE(191,48)-(215,23),PSET,
BF
360 FOR K=0 TO 9
370 COLOR 2:LINE(26,50+K*10)-(3
0,50+K*10),PSET
380 NEXT
390 DRAW"BM20,46C4D6L4U6R4BD90N
L4D6L4U6BL4ND6L4D3R4C2"
400 FOR K=0 TO 9 STEP 3
410 LINE(225,50-K)-(229,50-K),P
SET
420 NEXT
430 DRAW"BM232,48C4D6R4U6L4BU9N
R4U3R4D6"
440 RETURN
```



```
10 SCREEN 0:COLOR 1,14,14
20 LOCATE 3,10:INPUT"Qual o cur
so do êmbolo(1-90)";C
30 IF C<1 OR C>90 THEN 20
40 GOSUB 300
50 FOR I=0 TO C-1
60 LINE(31,37+I)-(44,50+I),14,B
F
70 LINE(34,50+I)-(39,43+I),15,B
F
80 LINE(181,50-I/10)-(225,50-I/
10),2
90 LINE(191,23-I/10)-(215,48-I/
10),15,BF
100 NEXT I
110 IF INKEY$="" THEN 110 ELSE
GOTO 10
300 SCREEN 2:COLOR 15,14,14
310 DRAW"BM30,30C2D131R195U131B
L45D20NR45D96L135U96NL15U20"
320 PAINT(200,70),2,2
330 DRAW"BM32,50C3R10BR140R41"
340 DRAW"BM127,161C2U20BH8C4R16
NH5G5"
350 LINE(34,48)-(39,41),,BF
360 LINE(191,48)-(215,23),,BF
370 FOR K=0 TO 9
380 LINE(26,50+K*10)-(30,50+K*1
0),2
390 NEXT K
400 DRAW"BM20,46C4D6L4U6R4BD90N
L4D6L4U6BL4ND6L4D3R4C2"
410 FOR K=0 TO 9 STEP 3
420 LINE(225,50-K)-(229,50-K)
430 NEXT K
440 DRAW"BM232,48C4D6R4U6L4BU9N
R4U3R4D6"
450 RETURN
```



```
20  HOME : HTAB (5): VTAB (10)
30  INPUT "QUAL O CURSO DO EMBO
LO(1-90)? ";C
40  GOSUB 300
50  FOR I = 1 TO C - 1
55  HCOLOR= 0
60  HPLOT 31,49 + I TO 44,49 +
I
70  HPLOT 34,42 + I TO 39,42 +
I
80  HCOLOR= 3
90  HPLOT 34,49 + I TO 39,49 +
I
100  HPLOT 181,50 - I / 10 TO 2
24,50 - I / 10
110  HPLOT 191,25 - I / 10 TO 2
15,25 - I / 10
120  NEXT I
130  FOR T = 1 TO 5000: NEXT T:
 TEXT : GOTO 20
300  HGR2 : HCOLOR= 3
310  HPLOT 30,30 TO 30,161 TO 2
25,161 TO 225,30
320  HPLOT 180,30 TO 180,146 TO
 45,146 TO 45,30
330  FOR I = 0 TO 96
340  HPLOT 30,50 + I TO 45,50 +
 I
350  HPLOT 180,50 + I TO 225,50
 + I
360  NEXT I
370  FOR I = 0 TO 15
380  HPLOT 30,146 + I TO 225,14
6 + I
390  NEXT I
400  FOR I = 0 TO 9
410  HPLOT 26,50 + I * 10 TO 30
,50 + I * 10
420  NEXT I
430  FOR I = 0 TO 9 STEP 3
440  HPLOT 225,50 - I TO 229,50
 - I
450  NEXT I
460  HPLOT 20,46 TO 20,53 TO 15
,53 TO 15,46 TO 19,46
470  HPLOT 11,140 TO 7,140 TO 7
,136 TO 11,136 TO 11,143 TO 7,1
43
480  HPLOT 20,136 TO 20,143 TO
15,143 TO 15,136 TO 20,136
490  HPLOT 232,48 TO 237,48 TO
237,55 TO 232,55 TO 232,48
500  HPLOT 232,42 TO 237,42 TO
237,35 TO 232,35 TO 232,39 TO 2
37,39
510  HPLOT 103,140 TO 117,140 T
O 110,136: HPLOT 117,140 TO 110
,144
530  FOR J = 1 TO 14
540  HPLOT 34,43 + J TO 39,43 +
 J
550  NEXT J
560  FOR K = 1 TO 26
570  HPLOT 191,24 + K TO 215,24
 + K
580  NEXT K
590  RETURN
```



A alavanca na tela do TRS-Color.

Inicialmente, o computador pergunta quanto você quer que o êmbolo desça, em um intervalo de 1 a 90. A rotina que começa na linha 300 desenhará o elevador e o laço da linha 110 (na versão para os micros Spectrum e TRS-Color) ou da 50 (MSX, Apple e TK-2000) deverá movimentá-lo.

PRINCÍPIOS

Quanto maior for o diâmetro do êmbolo maior terá que ser a força aplicada. Na nossa simulação, ele é bem menor que o diâmetro do elevador; portanto, a força para empurrá-lo para baixo será bem menor que o peso do objeto a ser levantado. Observamos, porém, que, assim como no caso das polias, o trajeto do êmbolo é maior que a altura da carga — ou seja, teremos que mover o êmbolo em várias unidades para que o elevador suba uma só.

CONTROLE POR TECLAS MÚLTIPLAS

No controle de jogos, o joystick ganha em sofisticação, enquanto o teclado se destaca pela versatilidade. Veja como fazer para que sejam detectadas várias teclas ao mesmo tempo.

Uma das características mais importantes num jogo é a qualidade de interação computador-usuário. Na maioria dos casos, o joystick oferece um controle mais sofisticado, mas o teclado o supera, sem dúvida alguma, no que diz respeito à versatilidade.

Em geral, utilizam-se os comandos **INKEY$** e **GET$** para detectar se uma tecla está sendo pressionada. Esses comandos, porém, apresentam uma deficiência: só são capazes de detectar duas teclas ao mesmo tempo caso uma delas seja **SHIFT** (ou similares, como **CTRL** ou **SYMBOL/SHIFT**).

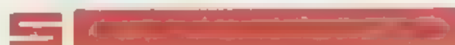
A detecção de tais combinações é suficiente para a maior parte dos propósitos, mas não para todos. Se quiséssemos, por exemplo, controlar os movimentos vertical e horizontal de uma nave, disparar o laser e lançar bombas, simultaneamente, seríamos obrigados a acionar quatro teclas. A solução para esse tipo de problema depende do próprio computador.

EFEITO DO ACIONAMENTO DAS TECLAS

Para entender como se dá a detecção múltipla, precisamos examinar os mecanismos que permitem ao computador saber qual tecla está sendo pressionada. Os micros pessoais utilizam, comumente, dois processos. O primeiro consiste em "varrer" o teclado de tempos em tempos, verificando se alguma tecla foi pressionada. No TRS-Color, por exemplo, o teclado é "varrido" a cada 1/100 de segundo; já no MSX, isso ocorre a cada 1/60 de segundo.

No segundo processo, o acionamento de uma tecla gera uma mensagem — chamada *interrupção* —, que é enviada ao processador. O computador interrompe sua tarefa e verifica o teclado, em busca da tecla pressionada. Esse método é bem mais eficiente e versátil que o da varredura, porque dispensa a verificação das teclas quando não estão sendo acionadas — ou seja, o primeiro método varre o teclado de tempos em tempos, mesmo que nenhuma tecla seja acionada; o segundo só faz a varredura quando há acionamento de teclas.

Seja qual for o método utilizado, o computador deve ser capaz de identificar a tecla acionada o mais rapidamente possível. A maioria dos teclados emprega um sistema denominado *gerador de matriz*, fornecendo ao computador um número que identifica a posição da tecla na matriz. O que vai ocorrer depois da geração do número varia de máquina para máquina.



O teclado do Spectrum compõe-se de quatro fileiras com dez teclas cada. Supondo que cada fileira seja dividida ao meio, teremos oito fileiras com grupos de cinco teclas. Cada grupo, por sua vez, tem comunicação com uma porta de entrada/saída. O teclado do Spectrum facilita a detecção de várias teclas acionadas ao mesmo tempo, pois possui 65536 dessas portas, identificadas por números de 0 a 65535. A tabela a seguir fornece o endereço da porta para cada um dos oito grupos:

TECLAS	GRUPO	PORTA
V C X Z CAPS/SHIFT	0	65276
G F D S A	1	65022
Q W E R T	2	64510
5 4 3 2 1	3	63486
6 7 8 9 0	4	61438
Y U I O P	5	57342
H J K L ENTER	6	49150
B N M SYM/SHIFT ESPAÇO	7	32766

Para calcular o endereço da porta de cada grupo, use esta fórmula:

$254 + 256*(255 - 2\uparrow n)$

Nessa fórmula, n significa o número do grupo listado na tabela.

Digite e rode o programa a seguir. Ele calcula o endereço da porta para um certo grupo de teclas.

```
10 INPUT "DIGITE NUMERO DO GR
UPO DA TECLA ";n
20 PRINT '"NUMERO DO GRUPO DA
 TECLA ";n
30 PRINT "ENDERECO DA PORTA "
;254+256*(255-2^n)
40 GOTO 10
```

Cada endereço de porta na tabela dada anteriormente assume um valor, dependendo da tecla pressionada.

Para usar corretamente o próximo programa, remova a impressora ou qualquer outra interface. Caso contrário, os resultados serão alterados. Digite estas linhas, rode, e acione de diferentes maneiras as teclas de 1 a 5, inclusive várias ao mesmo tempo:

```
10 PRINT AT 0,0;IN 63486
20 GOTO 10
```

Observe que os números exibidos na tela mudam conforme a combinação de



teclas acionadas. O valor no endereço da porta é armazenado em um único byte, no qual o quinto bit é sempre 1. Normalmente, o sexto e o sétimo bits são 0. Porém, quando houver um sinal no soquete EAR, ou quando o computador esquentar, o sexto bit passa a ser 1. Os bits de 0 a 4 representam as teclas do grupo. Cada bit assume valor 1 quando a tecla correspondente a ele não estiver sendo pressionada e valor 0, quando esta for pressionada. O quarto bit corresponde à tecla da extrema esquerda do grupo e o bit zero à tecla da extrema direita.

Quando um determinado bit contém o valor 1, ele contribui para o valor que está na porta; quando seu valor é 0, não contribui com nada. A contribuição de cada bit para o valor na porta está na tabela a seguir:

BIT	8	7	6	5	4	3	2	1
CONTRIBUIÇÃO	128	64	32	16	8	4	2	1

Portanto, se o sexto bit for 0 (nada conectado em EAR) e a tecla 5 estiver sendo pressionada, a disposição dos bits na porta 63486 será 00101111, o que equivale a 0+0+32+0+8+4+2+1=47. Assim, podemos detectar o acionamento das cinco teclas de um grupo. O mesmo se aplica aos outros grupos de teclas. Poderíamos, por exemplo, acrescentar ao programa linhas que imprimissem na tela o valor de cada porta.



T

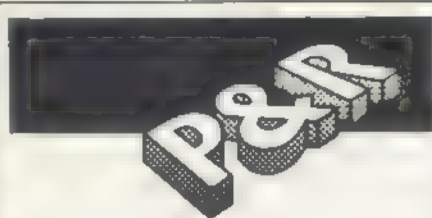
As teclas do TRS-Color estão dispostas numa matriz de sete linhas por oito colunas. São necessários nove bytes de memória para armazenar o estado de uma linha e de seus oito elementos. Dos nove bytes, o primeiro contém o estado da linha e os outros oito, o de cada um dos oito elementos dela. O arranjo da matriz e o método usado na decodificação da linha fazem com que, normalmente, seja impossível detectar mais de uma tecla ao mesmo tempo.

No entanto, podemos "enganar" o varredor de teclado, levando-o a não detectar nenhuma tecla numa certa linha, o que, na varredura seguinte, o obrigaria a assumir qualquer tecla como uma nova ocorrência. Para isso, é necessário colocar no primeiro byte o valor hexadecimal FF, produzindo uma alteração no estado das linhas.

Esse método não é muito satisfatório, pois requer freqüentes mudanças ao longo do programa. Mas há uma solução alternativa: chamar uma rotina em linguagem de máquina para forçar uma varredura completa do teclado, sempre que precisarmos. Como você verá, este foi o sistema usado nos programas que apresentaremos mais adiante.

No TK-2000, as teclas estão dispostas num gerador de matriz de oito linhas por seis colunas. Existem dois endereços de memória que trabalham com a matriz: o endereço −16384, ao qual demos o nome de KBOUT, e o −16368, que chamamos de KBIN. KBOUT controla o estado das linhas da matriz. Cada um de seus bits é responsável por uma linha na matriz de tecla. Se um bit de KBOUT possui valor 1, todos os elementos (teclas) da linha correspondente também serão iguais a 1.

KBIN, por sua vez, controla o esta-



O que é *roll-over*?

Esta é uma expressão inglesa que designa uma característica do processador de teclados de algumas linhas de microcomputadores: a possibilidade de armazenar em uma memória intermediária (*buffer* de teclado) as *n* últimas teclas pressionadas.

O número de teclas digitadas que o computador consegue armazenar depende do tamanho do buffer de teclado (se for fixado em hardware) e do software de processamento do teclado. Por isso, falamos em *n-key-roll-over*, ou seja, quantas teclas o roll-over é capaz de processar. Alguns computadores têm roll-over de duas a cinco teclas; outros, de até 128.

Qual é a vantagem de dispor de roll-over no teclado de um micro? Bem, certos softwares (sobretudo processadores de texto) não conseguem acompanhar um ritmo mais acelerado de digitação. Se não houver a capacidade de roll over, o computador acaba limitando o usuário, que começa a "perder" várias teclas pressionadas rapidamente, levando a erros irritantes de digitação. O roll-over afasta esse risco, pois, mais cedo ou mais tarde, o programa acaba "alcançando" o digitador.

do dos elementos de cada linha. Somente os bits de 0 a 5 de KBIN são usados, já que existem apenas seis elementos em cada linha. Os bits de KBIN só lêem o valor contido nas teclas correspondentes quando estas são pressionadas — ou seja, se nenhuma tecla estiver sendo pressionada, KBIN vale 0 (todos os bits são iguais a zero). Esse endereço requer uma atenção especial, uma vez que lê exclusivamente colunas. Se quisermos saber a que linha pertencem os elementos (teclas) lidos por KBIN, precisaremos identificar o valor contido em KBOUT.

Para forçar a varredura do teclado, foi necessário montar uma rotina em linguagem de máquina. Essa rotina, armazenada a partir do endereço 770 no primeiro programa, procura pelo acionamento de cinco teclas específicas: as quatro setas e a barra de espaço. A pressão sobre qualquer outra tecla não terá nenhum efeito.

A rotina usa os endereços 900 a 904 como indicadores do estado das teclas. Se o valor for 1, a tecla foi acionada; se for 0, não houve pressão sobre ela. Como se pode observar nas linhas 135 a 155, esses endereços são constantemente zerados. Isso é feito para a rotina não "pensar" que a tecla correspondente está sempre pressionada.

Remova o **POKE 900,0** da linha 135 e veja o que ocorre com a mira quando se aperta a tecla ↑. Ela começa a subir e não pára mais. Foi preciso usar endereços (900 a 904) como indicadores porque, coincidentemente, todas as cinco teclas em questão são lidas pelo mesmo bit de KBIN — ou seja, na matriz, elas estão em linhas diferentes mas em colunas iguais.

A rotina do segundo programa funciona de maneira semelhante, detectando seis teclas em vez de cinco. O endereço inicial foi deslocado para 36900 e os indicadores aparecem a partir de 36800. Essa rotina apresenta um aperfeiçoamento: os indicadores são zerados dentro dela própria.



A matriz que contém as teclas do MSX é maior que a dos outros micros: compõe-se de dez linhas com oito colunas cada, totalizando, assim, oitenta teclas. Nela existem duas portas de controle. A de endereço 170 ativa a linha que desejamos ler, enquanto a de endereço 169 contém o estado das teclas da linha ativada, sendo que cada tecla corresponde a um de seus bits.

Para simplificar, chamaremos a porta que ativa as linhas (170) de porta C, e a que lê as colunas, de porta B. A porta C usa somente os bits de 0 a 3 para ativar seqüencialmente cada uma das dez linhas. Isso é possível graças a um decodificador localizado entre a porta C e a matriz. As linhas da matriz de teclas são numeradas de 0 a 9. Para ativá-las, basta fornecer à porta C o número da linha desejada. Ao se ativar uma linha, todas as suas colunas (teclas) assumem o valor 1. Quando se pressiona uma tecla pertencente a uma linha ativada, o bit da porta B correspondente a essa tecla é zerado. Portanto, se ativarmos uma linha através da porta C, e, logo em seguida, lermos a porta B sem que nenhuma tecla seja acionada, veremos que a porta B tem valor 255, ou seja, todos os oito bits têm valor igual a 1.

O programa a seguir ativa a linha 8 da matriz — que contém as teclas das setas e barra de espaço — e, em seguida, lê a porta B, verificando quais bits foram zerados, isto é, quais teclas foram acionadas.



Os usuários do Apple devem ter notado a ausência de programas para esse micro. É que o método utilizado pelo Apple para detectar teclas torna quase impossível — dentro dos propósitos deste artigo — criar rotinas que detectem acionamento múltiplo de teclas.

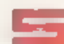
Os programas apresentados a seguir permitem-nos mover uma mira pela tela e também atirar. Nos micros TRS-Color, MSX e TK-2000, use as setas para mover e a barra de espaço para atirar. No Spectrum, use as setas para mover e o número 9 para atirar. Os programas do TRS-Color e do TK-2000 requerem um pequeno programa em linguagem de máquina para detectar as teclas.



```
10 CLEAR 200,32746
20 FOR K=32747 TO 32767:READ A:
POKE K,A:NEXT
30 DATA 48,140,9,191,1,155,134,
126,183,1,154,57,52,3,134,127,1
83,1,81,53,131
35 EXEC 32747
40 V=247:P=1295:L=1295:CLS3
50 IF PEEK(341)=V AND P>1055 TH
EN P=P-32
60 IF PEEK(342)=V AND P<1504 TH
EN P=P+32
70 IF PEEK(343)=V AND P>1024 TH
EN P=P-1
80 IF PEEK(344)=V AND P<1535 TH
EN P=P+1
90 IF PEEK(345)=V THEN SOUND 20
0,1
100 POKE L,175:POKE P,43:L=P:GO
TO 50
```

Digite e rode o programa do TRS-Color para se certificar de que não há erros. Em seguida, acrescente a linha 35 e rode-o novamente.

```
35 EXEC 32747
```



```
 10 BORDER 0: PAPER 0: INK 7
 20 CLS
 30 LET y=11: LET x=15
 40 LET p=63486
 50 GOSUB 220
 60 IF i=31 THEN  GOSUB 290
 70 LET p=61438
 80 GOSUB 220
 90 IF i=59 THEN  GOSUB 380
100 IF i=47 THEN  GOSUB 320
110 IF i=55 THEN  GOSUB 350
120 IF i=61 THEN  GOSUB 410
130 IF i=57 THEN  GOSUB 380:
GOSUB 410
140 IF i=45 THEN  GOSUB 320:
GOSUB 410
150 IF i=53 THEN  GOSUB 350:
GOSUB 410
160 IF i=43 THEN  GOSUB 380:
```

```
GOSUB 320
170 IF i=51 THEN  GOSUB 380:
GOSUB 350
180 IF i=41 THEN  GOSUB 380:
GOSUB 320: GOSUB 410
190 IF i=49 THEN  GOSUB 380:
GOSUB 350: GOSUB 410
200 GOSUB 250
210 GOTO 40
220 LET i=IN p
230 IF i>191 THEN  LET i=i-64
240 RETURN
250 PRINT AT y,x;" + "
260 PRINT AT y+1,x;"   "
270 PRINT AT y-1,x;"   "
280 RETURN
290 IF x<1 THEN  RETURN
300 LET x=x-1
310 RETURN
320 IF y>19 THEN  RETURN
330 LET y=y+1
340 RETURN
350 IF y<2 THEN  RETURN
360 LET y=y-1
370 RETURN
380 IF x>28 THEN  RETURN
390 LET x=x+1
400 RETURN
410 SOUND .004,20
420 SOUND .004,10
430 PRINT AT y,x;"   "
440 RETURN
```



```
10 COLOR 1,5,5:SCREEN2,1
20 X=120:Y=90:H=240:B=169:C=170
30 FOR K=1 TO 8:READ A
40 B$=B$+CHR$(A)
50 NEXT
60 DATA 129,66,60,36,36,60,66,1
29
70 SPRITE$(0)=B$
80 OUT C,(INP(C) AND H)OR 8
90 I=INP(B)
100 IF(I AND 128)=0 THEN X=X+3
110 IF(I AND 64)=0 THEN Y=Y+3
120 IF(I AND 32)=0 THEN Y=Y-3
130 IF(I AND 16)=0 THEN X=X-3
140 IF(I AND 1)=0 THEN PLAY"L64
BAC"
150 PUTSPRITE 0,(X,Y),1,0:GOTO
80
```



```
10  HOME
15  FOR K = 800 TO 804
20  READ A
25  POKE K,A
30  NEXT
35  DATA  64,32,16,8,4
40  FOR K = 770 TO 793
45  READ A
50  POKE K,A
55  NEXT
60  DATA  162,0,189,32,3,141
65  DATA  0,192,169,1,45,16
70  DATA  192,240,3,157,132,3
75  DATA  232,224,5,208,235,96
80  HGR2
85  H = 138:V = 96
90  X = 138:Y = 96
95  HCOLOR= 0
100  HPLOT H,V TO H + 4,V
105  HPLOT H + 2,V - 2 TO H + 2
 TO V + 2
110 H = X:V = Y
115  HCOLOR= 3
120  HPLOT X,Y TO X + 4,Y
125  HPLOT X + 2,Y - 2 TO X + 2
,Y + 2
130  CALL 770
135  IF  PEEK (900) = 1 AND Y >
 3 THEN Y = Y - 2: POKE 900,0
140  IF  PEEK (901) = 1 AND Y <
 188 THEN Y = Y + 2: POKE 901,0

145  IF  PEEK (902) = 1 AND X <
 274 THEN X = X + 2: POKE 902,0

150  IF  PEEK (903) = 1 AND X >
 1 THEN X = X - 2: POKE 903,0
155  IF  PEEK (904) = 1 THEN  P
RINT  CHR$ (7): POKE 904,0
160  GOTO 95
```

UM TECLADO A QUATRO MÃOS

Rodando o programa a seguir, você verá, logo após a introdução, dois seres vestindo "traje lunar", em posição de duelo. Este jogo foi feito para dois jogadores. No Spectrum, as teclas de comando são as seguintes: 1 move o ser da esquerda para cima, Q, para baixo, e A dispara o laser; 0, P e E controlam os movimentos do ser da direita. No TRS-Color, o jogador da esquerda usa as setas para cima e para baixo e a tecla Z; o da direita usa –, @ e /. No MSX, o da esquerda usa E, D e F; o da direita usa I, J e H. No TK-2000, finalmente, o jogador da esquerda usa 1, A e Q; o da direita, 0, (:) e P.



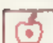
```
10 CLS:PMODE 4,1:SS=PEEK(186)*2
56+PEEK(187)
20 FOR K=SS TO SS+480 STEP 32
30 FOR J=K TO K+3:READ A:POKE J
,A:NEXT J,K
40 DATA 1,192,3,128,3,192,3,192
,6,0,0,96,15,192,3,240
50 DATA 31,192,3,248,63,192,3,2
52,15,0,0,240,15,135,113,240
60 DATA 15,248,15,240,15,248,15
,240,15,128,1,240,15,128,1,240
70 DATA 15,128,1,240,12,192,3,4
8,12,192,3,48,14,224,7,112
80 DIM L(6),R(6),B(6)
90 GET(0,0)-(15,15),L:GET(16,0)
-(31,15),R,G
100 PCLS,PRINT @10,"D U E L O"
110 PRINT @98,"GANHE PONTOS ACE
RTANDO O SEU      OPONENTE. CADA
 JOGADOR TEM                SEIS BA
LAS."
120 PRINT:PRINTTAB(6);"C O N T
R O L E S"
130 PRINT:PRINT" JOGADOR 1";TAB
(19);"JOGADOR 2"
140 PRINT:PRINT"   UP      --CIM
A--      -"
150 PRINT"  DOWN     --BAIXO-
  P"
160 PRINT"    Z      --FOGO--
  /"
170 PRINT @482,"qualquer tecla
para comecar";
180 IF INKEY$="" THEN 180
190 X1=16:X2=232:Y1=88:Y2=88:B1
=6:B2=6:S1=0:S2=0:PCLS
200 PUT(X1,Y1)-(X1+15,Y1+15),L,
PSET:PUT(X2,Y2)-(X2+15,Y2+15),R
,PSET
210 FOR K=1 TO 6:CIRCLE(10*K,2)
,1,5:CIRCLE(255-10*K,2),1,5:NEX
T
220 SCREEN 1,1:A$=INKEY$
230 L1=Y1:L2=Y2
240 IF PEEK(341)=247 AND Y1>16
THEN Y1=Y1-8
250 IF PEEK(342)=247 AND Y1<176
 THEN Y1=Y1+8
260 IF PEEK(343)=223 AND Y2>16
THEN Y2=Y2-8
270 IF PEEK(338)=251 AND Y2<176
 THEN Y2=Y2+8
280 IF B1=0 AND B2=0 THEN 360
290 IF L1=Y1 THEN 310
300 PUT (X1,L1)-(X1+15,L1+15),B
:PUT(X1,Y1)-(X1+15,Y1+15),L
310 IF L2=Y2 THEN 330
320 PUT(X2,L2)-(X2+15,L2+15),B:
PUT(X2,Y2)-(X2+15,Y2+15),R
330 IF PEEK(340)=247 GOSUB 1000
340 IF PEEK(345)=223 GOSUB 1500
350 GOTO 230
360 CLS:IF S1>S2 THEN PRINT @96
,"JOGADOR 1 GANHOU POR";S1;"A";
S2:GOTO 390
370 IF S2>S1 THEN PRINT @96,"JO
GADOR 2 GANHOU POR";S2;"A";S1:G
OTO 390
380 PRINT @96,"HOUVE EMPATE :";
S1;"A";S1;"!"
390 A$=INKEY$:GOTO 180
1000 IF B1=0 THEN RETURN
1010 PLAY"T12005AGFEDC"
1020 FOR N=32 TO 232 STEP 16
1030 LINE(N+1,Y1+7)-(N+6,Y1+7),
PSET
1040 LINE(N+1,Y1+7)-(N+6,Y1+7),
PRESET
1050 NEXT
1060 IF Y1=Y2 OR Y1+8=Y2 THEN S
1=S1+1:CIRCLE(10*S1,8),2,5:PLAY
"T801GDBC"
1070 CIRCLE(10*B1,2),1,0:B1=B1-
1
1080 RETURN
1500 IF B2=0 THEN RETURN
1510 PLAY"T12005BAGFEDC"
1520 FOR N=216 TO 32 STEP -16
1530 LINE(N+1,Y2+7)-(N+6,Y2+7),
PSET
1540 LINE(N+1,Y2+7)-(N+6,Y2+7),
PRESET
1550 NEXT
1560 IF Y2=Y1 OR Y2+8=Y1 THEN S
2=S2+1:CIRCLE(255-10*S2,8),2,5:
PLAY"T801GDBC"
```

```
1570 CIRCLE(255-10*B2,2),1,0:B2
=B2-1
1580 RETURN
```

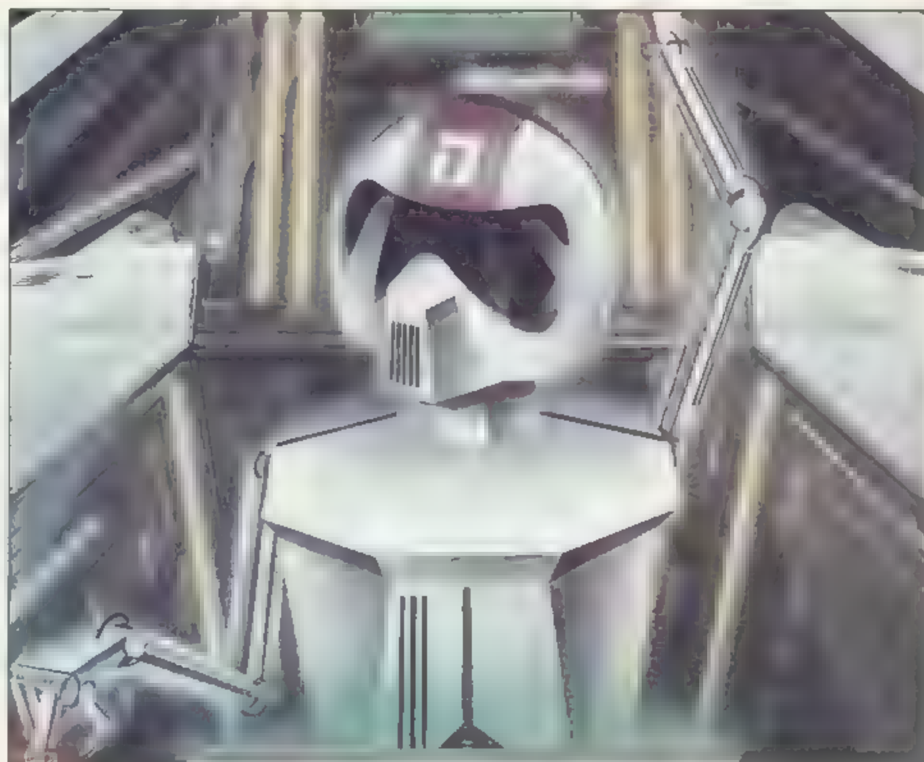
```
10 BORDER 0: PAPER 0: INK 7
20 BRIGHT 0: OVER 0: CLS
30 PRINT AT 1,9; INK 6; FLASH
 1;" D U E L O "
40 PRINT : PRINT
50 PRINT  INK 5;"  GANHE PONT
OS ACERTANDO SEU        OPONE
NTE. CADA JOGADOR
  TEM SEIS BALAS."
60 PRINT : PRINT
70 PRINT TAB 6; INVERSE 1;"C
O N T R O L E S"
80 PRINT : PRINT " JOGADOR 1
         JOGADOR 2 "
90 PRINT : PRINT "     1     -
-CIMA--      0      "
100 PRINT : PRINT "     Q     -
-BAIXO-      P    "
110 PRINT : PRINT "     A     -
-TIRO--      ENTER"
120 PRINT : PRINT : PRINT TAB
6;"QUE VENCA O MELHOR"
130 FOR n=USR "a" TO USR "i"+7
140 READ d
150 POKE n,d
160 NEXT n
170 DATA 1,3,6,15,31,63,15,15
180 DATA 192,192,0,192,192,192
,0,135
190 DATA 15,15,15,15,15,12,12,
14
200 DATA 248,248,128,128,128,
192,192,224
210 DATA 0,0,0,0,0,0,126,0
220 DATA 3,3,0,3,3,3,0,113
230 DATA 128,192,96,240,248,
252,240,240
240 DATA 15,15,1,1,1,3,3,7
250 DATA 240,240,240,240,240,
48,48,112
260 LET y1=10: LET y2=10
270 LET b1=6: LET b2=6
280 LET s1=0: LET s2=0
290 RESTORE 330: FOR n=1 TO 8
300 READ d,p
310 SOUND d,p
320 NEXT n
330 DATA .1,7,.09,12,.1,7,.09,
12,.6,7,.45,2,.45,6,.5,0
340 PRINT #1;AT 0,0; FLASH 1;"
 QUALQUER TECLA PARA COMECAR
 "
350 LET p=254: GOSUB 570
360 IF i=191 THEN  GOTO 350
370 INK 4: BRIGHT 1: CLS
380 PRINT  INVERSE 1;"JOGADOR1
 0            0 JOGADOR2"
390 PRINT "BALAS :  6
   6  : BALAS"
400 PRINT #1;AT 0,0; INVERSE 1
;
410 GOSUB 600
420 LET p=63486: GOSUB 570
430 IF i=62 OR i=30 THEN
GOSUB 810
440 LET p=61438: GOSUB 570
450 IF i=62 OR i=30 THEN
GOSUB 840
460 LET p=64510: GOSUB 570
470 IF i=62 OR i=30 THEN
GOSUB 870
480 LET p=57342: GOSUB 570
490 IF i=62 OR i=30 THEN
GOSUB 900
500 LET p=49150: GOSUB 570
510 IF i=62 OR i=30 THEN
GOSUB 1020
520 LET p=65022: GOSUB 570
530 IF i=62 OR i=30 THEN
GOSUB 930
540 GOSUB 600
550 IF b1=0 AND b2=0 THEN
GOTO 1110
560 GOTO 420
570 LET i=IN p
580 IF i>191 THEN  LET i=i-64
590 RETURN
600 PRINT AT y1,1;CHR$ 144;
CHR$ 145
610 PRINT AT y1+1,1;CHR$ 146;
CHR$ 147
620 PRINT AT y2,29;CHR$ 149;
CHR$ 150
630 PRINT AT y2+1,29;CHR$ 151;
CHR$ 152
640 PRINT AT y1-1,1;"  "
650 PRINT AT y1+2,1;"  "
660 PRINT AT y2-1,29;"  "
670 PRINT AT y2+2,29;"  "
680 PRINT AT 0,9; PAPER 4; INK
9;s1
690 PRINT AT 0,22; PAPER 4;
INK 9;s2
700 PRINT AT 1,9;b1
710 PRINT AT 1,22;b2
720 RETURN
730 PRINT AT 10,10;"AAGH! ME A
CERTOU !"
740 RESTORE 780: FOR n=1 TO 11
750 READ d,p
760 SOUND d,p
770 NEXT n
780 DATA .5,2,.4,2,.2,2,.5,2,.
3,5,.2,4,.4,4,.2,2,.4,2,.2,1,.
5,2
790 PRINT AT 10,10;"
     "
800 RETURN
810 IF y1<4 THEN  RETURN
820 LET y1=y1-1
830 RETURN
840 IF y2<4 THEN  RETURN
850 LET y2=y2-1
860 RETURN
870 IF y1>18 THEN  RETURN
880 LET y1=y1+1
890 RETURN
900 IF y2>18 THEN  RETURN
910 LET y2=y2+1
920 RETURN
930 IF b1=0 THEN  RETURN
940 SOUND .01,4: SOUND .01,0
950 FOR n=3 TO 27
960 PRINT AT y1,n;" ";CHR$ 148
970 NEXT n
980 PRINT AT y1,27;"  "
990 IF y1=y2 OR y1=y2+1 THEN
LET s1=s1+1: GOSUB 730
1000 LET b1=b1-1
1010 RETURN
1020 IF b2=0 THEN  RETURN
1030 SOUND .01,0: SOUND .01,-10
1040 FOR n=27 TO 3 STEP -1
1050 PRINT AT y2,n;CHR$ 148;" "
1060 NEXT n
1070 PRINT AT y2,3;" "
1080 IF y2=y1 OR y2=y1+1 THEN
LET s2=s2+1: GOSUB 730
1090 LET b2=b2-1
1100 RETURN
1110 IF s1>s2 THEN  PRINT AT 10
,5; FLASH 1;" JOGADOR 1 VENCEU
!"
1120 IF s2>s1 THEN  PRINT AT 10
,5; FLASH 1;" JOGADOR 2 VENCEU
!"
1130 IF s1=s2 THEN  PRINT AT 10
,10; FLASH 1;" EMPATE ! "
1140 GOTO 260
```

```
10 GOTO 170
15 S1=S1-1:LINE(254,Y1+5)-(20,Y
1+5),15:LINE(254,Y1+5)-(20,Y1+5
),1
20 IFY1>Y2-5ANDY1<Y2+13THENP1=P
1+1:PUTSPRITE2,(X2,Y2),6,2
25 FORT=0TO50:NEXT:RETURN
30 S2=S2-1:LINE(0,Y2+5)-(235,Y2
+5),15:LINE(0,Y2+5)-(235,Y2+5),
1
35 IFY2>Y1-5ANDY2<Y1+13THENP2=P
2+1:PUTSPRITE1,(X1,Y1),6,2
40 FORT=0TO50:NEXT:RETURN
45 CLS:COLOR 15,2,2
50 LOCATE12,2:PRINT"DUELO ESTEL
AR"
55 PRINT:PRINT"ganhe pontos ace
rtando o seu oponente"
60 PRINTTAB(5)"cada jogador tem
 seis balas"
65 PRINT:PRINT:PRINT"jogador 1"
;SPC(19);"jogador 2":PRINT
70 PRINTTAB(3)"E  -------   cim
a  -------  I"
75 PRINTTAB(3)"D  -------   baix
o  -------  J"
80 PRINTTAB(3)"F  -------   fog
o  -------  H"
85 LOCATE10,20:PRINT"<qualquer
tecla>"
90 IF INKEY$="" THEN 90
95 COLOR 15,1,1:SCREEN 2,1
100 SPRITE$(0)=A$
105 SPRITE$(1)=B$
110 SPRITE$(2)=C$
115 IF S1=0 OR S2=0 THEN 225
120 OUT C,(INP(C)ANDH)OR3
125 I=INP(B)
130 IF(IAND2)=0ANDY1<176THEN Y1
=Y1+2
135 IF(IAND4)=0ANDY1>1THEN Y1=Y
1-2
140 IF(IAND8)=0 THEN GOSUB15
145 IF(IAND32)=0 THEN GOSUB30
150 IF(IAND64)=0ANDY2>1THEN Y2=
Y2-2
155 IF(IAND128)=0ANDY2<176THEN
Y2=Y2+2
160 PUTSPRITE1,(X1,Y1),15,0:PUT
SPRITE2,(X2,Y2),15,1
```

```
165 GOTO 115
170 S1=6:S2=6:P1=0:P2=0:C=170:H
=240
175 X1=5:X2=230:Y1=96:Y2=96:B=1
69
180 FOR K=1 TO 8
185 READ P,Q,R
190 A$=A$+CHR$(P)
195 B$=B$+CHR$(Q)
200 C$=C$+CHR$(R)
205 NEXT K
210 DATA 48,12,129,48,12,66,255
,255,36,112,14,24
215 DATA 48,12,24,40,20,36,72,1
8,66,144,9,129
220 GOTO 45
225 SCREEN0:CLS
230 CLS:PRINT"FIM DE JOGO":PRIN
T
235 PRINT:PRINT"jogador 1 = ";P
1" pontos"
240 PRINT"jogador 2 = ";P2;" po
ntos"
245 LOCATE8,20:PRINT"JOGA DE NO
VO (S/N) ?"
250 I$=INKEY$:IFI$="S"THEN RUN
255 IF I$<>"N"THEN 250
260 CLS:COLOR15,4,4:END
```



```
10 P = 36800:Q = 32:S1 = 6:S2 =
 6:P1 = 0:P2 = 0:X1 = 1
15 X2 = 271:Y1 = 96:Y2 = 96:H1
= 1:H2 = 271:V1 = 96:V2 = 96
20  GOTO 85
25 S1 = S1 - 1
30  HPLOT 20,Y1 TO 279,Y1: HCOL
OR= 0
35  HPLOT 20,Y1 TO 279,Y1: HCOL
OR= 3
40  IF Y1 > Y2 - 3 AND Y1 < Y2
+ 6 THEN  GOTO 75
45  RETURN
50 S2 = S2 - 1
55  HPLOT 260,Y2 TO 0,Y2: HCOLO
R= 0
60  HPLOT 260,Y2 TO 0,Y2: HCOLO
R= 3
65  IF Y2 > Y1 - 3 AND Y2 < Y1
+ 6 THEN  GOTO 80
70  RETURN
75 P1 = P1 + 1: PRINT  CHR$ (7)
: RETURN
80 P2 = P2 + 1: PRINT  CHR$ (7)
: RETURN
85  HOME :E = 35000: HIMEM: E
90 F =  INT (E / 256): POKE 232
,E - F * 256: POKE 233,F
95  FOR I = E TO E + 41 + 2 * 3
2
100  READ A: POKE I,A
105  NEXT
110  SCALE= 1: ROT= 0
115  DATA    20 ,0 ,42 ,0 ,74 ,
0 ,106 ,0 ,138 ,0 ,170 ,0 ,202
,0 ,234 ,0 ,10 ,1 ,42 ,1 ,74 ,1
 ,106 ,1 ,138 ,1 ,170 ,1 ,202 ,
1 ,234 ,1 ,10 ,2 ,42 ,2 ,74 ,2
,106 ,2 ,138,2
120  DATA   0 ,72 ,109 ,73 ,218
 ,219 ,255 ,42 ,45 ,45 ,45 ,213
 ,219 ,59 ,191 ,9 ,109 ,73 ,218
```



```
 ,27 ,31 ,159 ,105 ,105 ,137 ,2
19 ,27 ,223 ,6 ,0 ,0 ,0
125  DATA   0 ,72 ,9 ,109 ,209
,27 ,255 ,155 ,45 ,45 ,45 ,173
,27 ,63 ,223 ,83 ,73 ,109 ,209
,27 ,31 ,223 ,74 ,105 ,105 ,26
,223 ,223 ,19 ,0 ,0 ,0
130  REM  SUBROTINA DE LEITURA
135  REM      DO TECLADO
140  FOR K = 36810 TO 36815
145  READ A: POKE K,A: NEXT
150  DATA  64,32,16,8,4,2
155  FOR K = 36900 TO 36935
160  READ A: POKE K,A: NEXT
165  DATA     162,6,169,0,157,19
1,143,202,224,0,208,248,162,0,1
89,202,143,141
170  DATA   0,192,169,32,45,16,
192,240,3,157,192,143,232,224,6
,208,235,96
175  VTAB (4): INVERSE : PRINT
 TAB( 39)" ": NORMAL
180  PRINT  TAB( 15)" D U E L O
"
185  INVERSE : PRINT  TAB( 39)"
 ": NORMAL
190  PRINT : PRINT " GANHE PONT
OS ACERTANDO O SEU OPONENTE"
195  PRINT  TAB( 6)"CADA JOGADO
R TEM SEIS BALAS"
200  PRINT : PRINT : PRINT "JOG
ADOR 1 ------------------- JOGA
DOR 2": PRINT
205  PRINT "   1    -----------CI
MA----------    0"
210  PRINT "    Q    ----------FO
GO----------    P"
215  PRINT "    A    ---------BAI
XO----------    :"
220  PRINT : PRINT : PRINT : PR
INT
225  HTAB (13): INVERSE : PRINT
 "QUALQUER TECLA": NORMAL
230  GET Z$: IF Z$ = "" THEN 23
0
235  HGR2 : HCOLOR= 3
240  DRAW 1 AT X1,Y1: DRAW 2 AT
 X2,Y2
245  XDRAW 1 AT H1,V1: XDRAW 2
AT H2,V2
250 V1 = Y1:V2 = Y2
255  DRAW 1 AT X1,Y1: DRAW 2 AT
 X2,Y2
260  CALL 36900
265  IF S1 = 0 OR S2 = 0 THEN
GOTO 305
270  IF  PEEK (P) = Q AND Y2 <
181 THEN Y2 = Y2 + 3
275  IF  PEEK (P + 1) = Q THEN
 GOSUB 50
280  IF  PEEK (P + 2) = Q AND Y
2 > 10 THEN Y2 = Y2 - 3
285  IF  PEEK (P + 3) = Q AND Y
1 > 10 THEN Y1 = Y1 - 3
290  IF  PEEK (P + 4) = Q THEN
 GOSUB 25
295  IF  PEEK (P + 5) = Q AND Y
1 < 181 THEN Y1 = Y1 + 3
300  GOTO 245
305  TEXT : HOME
310  PRINT "FIM DE JOGO": PRINT
 : PRINT
315  PRINT "JOGADOR 1 = ";P1;"
PONTOS"
320  PRINT "JOGADOR 2 = ";P2;"
PONTOS"
325  PRINT : PRINT : INPUT "JOG
A DE NOVO (S/N) ?";A$
330  IF A$ = "S" THEN  RUN
335  END
```

OS SEGREDOS DO TRS-80 (3)

- MAIS USOS PARA VTRANSF
- CÓPIA DA TELA EM FITA OU DISCO
- COMO RECUPERAR UMA TELA GRAVADA

Como vimos em artigos anteriores, podemos utilizar vários "truques" para explorar melhor os recursos de vídeo do TRS-80. Aprenderemos agora a gravar uma tela em fita ou disco.

A sub-rotina **VTRANSF**, apresentada no segundo artigo desta série, será usada aqui por um programa que copia integralmente o conteúdo da tela de vídeo, armazenando-o, depois, em um arquivo de fita ou disco. Um outro programa permitirá a realização da operação inversa, ou seja, colocar novamente na tela um desenho ou texto armazenado em fita ou disco.

O método utilizado para isso é semelhante ao que vimos no artigo anterior, quando tratamos da impressão, linha por linha, do conteúdo da tela. A diferença é que, em vez de imprimirmos a variável **V$**, simplesmente a enviamos para fita ou disco.

ARMAZENAGEM EM FITA

Listamos, a seguir, a versão para fita. Depois de digitá-la, acrescente a subrotina 1000, apresentada no artigo anterior.

```
10 CLEAR 500
20 CLS : PRINT "PREPARE O GRAV
ADOR"
25 PRINT "E PRESSIONE <ENTER> "
30 C%=64 : A$=CHR$(34)
35 IF INKEY$="" THEN 35
40 FOR V%=0 TO 960 STEP 64
50 GOSUB 1000:PRINT #-1,A$;V$;
A$
60 NEXT V%
70 END
```

A linha 30 determina o comprimento da linha de vídeo a ser copiada (54 posições). Ela se encarrega, também, de definir a variável **A$**, que conterá o caractere "aspas". Este será usado na linha 50, para englobar a variável **V$**. A linha 35 espera que o usuário prepare o gravador e pressione a tecla **<ENTER>**. Só então o programa continua. O laço que vai da linha 40 à linha 60 copia as dezesseis linhas da tela, sucessivamente, na variável **V$**, usando a função **VARPTR**, e, em seguida, envia para o gravador 1.

É muito importante colocar todo o *string* **V$** entre aspas, pois, se houver alguma vírgula ou ponto e vírgula na tela, a transmissão para a fita será truncada, provocando um erro de leitura, posteriormente.

ARMAZENAGEM EM DISCO

A versão para disco é mais simples:

```
10 CLEAR 500
20 OPEN "O",1,"TELA1/VID"
30 C%=64
40 FOR V%=0 TO 960 STEP 64
50 GOSUB 1000 : PRINT #1,V$
60 NEXT V%
70 CLOSE 1:END
```

A linha 20 abre um arquivo de acesso seqüencial para saída, chamado TELA1/VID (poderia ser qualquer nome). As linhas 30 a 60 funcionam como na versão para fita, só que não são necessárias as aspas em torno de **V$**. A linha 70 fecha o arquivo criado.

LEITURA EM FITA

O programa que lê o conteúdo de tela no arquivo, trazendo-o de volta ao vídeo, obedece à mesma seqüência:

```
10 CLEAR 500
20 CLS : PRINT "PREPARE O GRAV
ADOR"
25 PRINT "E PRESSIONE <ENTER> "
30 C%=64
35 IF INKEY$="" THEN 35
40 FOR V%=0 TO 960 STEP 64
50 GOSUB 1000 : INPUT #-1,X$
55 V$=LEFT$(X$,C%)
60 NEXT V%
70 END
```

Observe que a sub-rotina **VTRANSF** também serve para o programa de leitura, sem modificações, já que sua função consiste em determinar a localização e tamanho do *string* **V$** na memória. Assim, depois de definida pela linha 55, a variável **V$** é automaticamente armazenada na memória de vídeo. Sua exibição na tela também é instantânea.

A função **LEFT$** toma apenas os 64 primeiros caracteres de **X$**, uma vez que, na gravação, um caractere ASCII 13 (*linefeed*) foi adicionado ao final de **V$**. Esse caractere prejudicaria toda a tela se não fosse retirado.

LEITURA EM DISCO

A versão do programa de leitura em disco é a seguinte:

```
10 CLEAR 500
20 OPEN "I",1,"TELA1/VID"
30 C%=64
40 FOR V%=0 TO 960 STEP 64
50 GOSUB 1000:LINE INPUT #1,X$
55 LSET V$=X$
60 NEXT V%
70 CLOSE 1:END
```

O **LSET** da linha 55 existe apenas no BASIC para disco. Tem a mesma função do **LEFT$** da versão para fita mas, em termos operativos, é mais simples que esse comando.

APLICAÇÕES

Existem muitas aplicações para os programas listados neste artigo. Uma das mais comuns é na montagem das telas de instruções de um jogo. Estas, quando mais elaboradas, podem incluir gráficos, juntamente com textos explicativos — por exemplo, a reprodução do desenho de cada tipo de nave inimiga e o número de pontos que o jogador ganha se abatê-las. Em vez de colocar todos esses dados dentro de uma mesma listagem, tornando-a longa e complicada, use os programas de transferência para jogar as figuras e o texto diretamente da fita ou disco para o vídeo do microcomputador, gastando apenas 64 bytes de memória RAM!

Os programas educativos constituem uma outra área importante de aplicação. As telas podem compor, por exemplo, as páginas de um "livro eletrônico". Armazenadas em disco, serão "folheadas" pelo aluno, que copiará aquelas que julgar necessário.

AVALANCHE: ACERTO DAS VARIÁVEIS

- O AVANÇO DA MARÉ
- GAIVOTAS, NUVENS E VENTO
- PEDRAS E COBRAS
- SITUAÇÃO DO PERSONAGEM



Uma das tarefas mais difíceis na programação de jogos em código de máquina é garantir a sincronia dos diversos eventos. Aqui estão as rotinas que cuidam disso.

Sempre que vamos recomeçar o jogo, precisamos acertar o escore e, também, estabelecer os parâmetros que influenciam o comportamento de outras sub-rotinas. Temos que esvaziar a maré que ameaçava Willie, definir a direção do vento, colocar Willie de volta ao pé do monte e, para que tudo ocorra no momento adequado, acertar os diversos laços controladores de tempo.

A rotina que se segue acerta o valor das diversas variáveis para que Willie possa se dedicar à árdua tarefa de recuperar seu lanche roubado.

```
10  REM org 58606
20 REM dth ld a,6
30 REM ld (57353),a
40 REM ld hl,736
50 REM ld (57354),hl
60 REM ld hl,130
70 REM ld (57345),hl
80 REM ld a,3
90 REM ld (57347),a
100 REM ld a,0
110 REM ld (57348),a
120 REM ld a,2
130 REM ld (57349),a
140 REM ld hl,449
150 REM ld (57332),hl
160 REM ld hl,0
170 REM ld (57334),hl
```

```
180 REM ld a,0
190 REM ld (57336),a
200 REM ld hl,223
210 REM ld (57356),hl
220 REM ld a,0
230 REM ld b,5
240 REM ld (57350),a
250 REM add a,b
260 REM ld (57351),a
270 REM add a,b
280 REM ld (57352),a
```

Essa rotina não é chamada apenas no início do jogo, mas, também, quando ele recomeça, depois de Willie encontrar a morte dentro de um buraco, afogado ou embaixo de alguma pedra.

O MAR

Colocando todas as variáveis juntas na memória do microcomputador, podemos verificar o status do jogo a qualquer instante, o que facilita muito a tarefa de depuração de erros.

Variáveis de um byte são modificadas através do acumulador, que tem oito bits, enquanto o par de registros HL se encarrega das variáveis de dois bytes, mesmo que o seu conteúdo no momento caiba em apenas um byte. Isso ocorre porque o byte mais significativo também deve ser modificado.

O endereço 57353 contém o atraso do movimento da maré. Colocamos ali o número 6, dando a Willie uma chance razoável de escalar a montanha antes de se afogar. Depois, esse valor poderá ser modificado para acelerar o avanço da maré, o que tornará o jogo mais difícil e mais emocionante.

O mar deve ser desenhado a partir da base da tela, a cada nova etapa do jogo. A posição da tela correspondente ao canto superior esquerdo do mar é colocada nos endereços 57354 e 57355. O número 736, valor inicial dessa variável, equivale ao endereço da extremidade inferior esquerda da tela.

A NUVEM

Na versão do jogo para o Spectrum, há uma nuvem cruzando o céu. O endereço 57345 contém sua posição de impressão na tela. O valor inicial dessa variável é 130.

Mas o programa utiliza outros dados para cuidar do movimento da nuvem. Se não provocarmos um atraso, ela atravessará o céu tão rapidamente quanto um avião a jato. Para evitar isso, colocamos um contador de atraso na posição 57347. Seu valor inicial é 3.

Precisamos também saber em que di-



reção sopra o vento, para que a nuvem se mova de acordo. A informação fica armazenada em 57348. Quando esse byte vale zero, a nuvem se movimenta para a direita; quando vale 1, para a esquerda. No programa, essa variável tem valor inicial zero, movendo, portanto, a nuvem para a direita.

VIDA OU MORTE

Um atraso para o movimento das gaivotas é colocado em 57349, com valor inicial 2. A posição que Willie ocupa na tela fica em 57352. O programa coloca ali o valor 449, correspondente ao canto inferior esquerdo do perfil da encosta da montanha.

Uma outra variável informa ao programa se nosso herói está parado, correndo ou saltando. Por motivos que conheceremos mais adiante, utilizaremos dois bytes, 57334 e 57335. Como Willie começa o jogo em pé, parado no sopé da montanha, o programa coloca zero naquelas posições.

A situação atual do jogo pode ser monitorada pela variável que fica em 57336. Se esse byte for 0, Willie vai bem, obrigado. Se for 1, ele acaba de recuperar parte do seu lanche e as rotinas do próximo nível de dificuldade vão ser chamadas. Se for 2, Willie morreu! Como no início do jogo nosso personagem está vivo, o programa coloca 0 nessa posição.

PEDRAS E COBRAS

A variável que controla a posição da pedra fica em 57356. O programa coloca ali seu valor inicial, 223, que corresponde ao topo da montanha, no canto superior direito.

As três cobras têm língua que se mexe. Como não é desejável que as línguas façam o mesmo movimento, precisamos provocar uma defasagem em seu ritmo. Para isso, colocam-se variáveis de atraso diferentes para cada língua, nos endereços 57350, 57351 e 57352. O acumulador recebe o valor zero, e o registro B, o número 5. Na primeira variável de atraso coloca-se, então, zero. Soma-se o conteúdo de B ao acumulador e o resultado, 5, é colocado no segundo atraso. Mais uma vez 5 é somado ao acumulador e o resultado, 10, vai para o terceiro atraso.



O programa em Assembly listado a seguir estabelece o valor inicial de uma série de variáveis que são usadas para controlar o jogo.

```
10   ORG 19447
20   NLV LDA #6
30   STA 18246
40   LDX #7424
50   STX 18247
60   LDX #5088
70   STX 18249
80   CLR 18251
90   CLR 18252
100   LDX #3070
110   STX 18253
120   CLR 18255
130   LDA 5
140   STA 18256
150   LDA #10
160   STA 18257
170   RTS
```

Essa rotina, que recebeu o rótulo **NLV**, é utilizada no início do jogo e, também, quando Willie morre.

O MAR

Uma das vantagens de se colocar todas as variáveis em um mesmo local da memória é que podemos verificar facilmente o status do jogo quando estamos depurando os erros.

O valor de variáveis de um byte é estabelecido com a ajuda do acumulador, que tem oito bits, enquanto o registro X, de dezesseis bits, se encarrega das variáveis de dois bytes, mesmo que o seu conteúdo, no momento, caiba em um único byte. Isso ocorre porque o byte mais significativo também deve ser modificado.

O endereço 18246 contém o atraso do movimento da maré. O programa coloca o valor 6 nessa posição para que Willie tenha uma chance razoável de subir a montanha antes de se afogar. Depois, poderemos mudar esse valor para acelerar o avanço da maré, tornando o jogo mais difícil e emocionante.

O mar deve ser desenhado a partir da base da tela, a cada nova etapa do jogo. A posição da memória de vídeo correspondente ao canto superior esquerdo do mar é colocada nos bytes 18247 e 18248. O programa guarda ali o valor inicial 7424, que corresponde ao canto inferior esquerdo da tela.

VIDA OU MORTE

A posição de Willie na tela fica armazenada em 18249. O programa coloca nesse byte o valor inicial 5088, correspondente ao canto inferior esquerdo do perfil da encosta da montanha, de

onde nosso herói deve partir em busca do lanche perdido

Uma outra variável, que fica em 18251, verifica se Willie está parado, andando ou pulando. O programa coloca valor zero nessa posição já que no início do jogo o personagem está parado, ao pé da montanha.

A situação atual do jogo pode ser monitorizada pela variável que fica em 18252. Se esse byte for 0, Willie vai bem, obrigado. Se for 1, ele acaba de recuperar parte do seu lanche e as rotinas do próximo nível de dificuldade vão ser chamadas. Se for 2, Willie morreu! Como no início do jogo nosso personagem está vivo, o programa coloca 0 nessa posição.

PEDRAS E COBRAS

A variável que controla a posição da pedra fica em 18253. O programa coloca ali seu valor inicial, 3070, que corresponde ao topo da montanha, no canto superior direito.

As três cobras têm língua que se mexe. Como não é desejável que as línguas façam o mesmo movimento, precisamos provocar uma defasagem em seu ritmo. Para isso, colocam-se variáveis de atraso diferentes para cada língua, nos endereços 18255, 18256 e 18257. A primeira variável de atraso é limpa com **CLR**, tornando-se zero. Os números 5 e 10 são colocados, respectivamente, na segunda e na terceira variáveis'de atraso, com o auxílio do acumulador.

A rotina listada a seguir ajusta uma série de variáveis com os valores que elas precisam conter quando Willie inicia sua arriscada missão.

```
10    org -11645
20    mrt ld a,6
30    ld (-5213),a
40    ld hl,736
50    ld (-5212),hl
60    ld hl,130
70    ld (-5210),hl
80    ld a,3
90    ld ( 5208),a
100   ld a,0
110   ld (-5207),a
120   ld a,2
130   ld (-5206),a
140   ld hl,481
150   ld (-5205),hl
160   ld hl,0
170   ld (-5203),hl
180   ld a,0
190   ld (-5201),a
200   ld hl,255
210   ld (-5200),hl
220   ld a,0
230   ld b,5
240   ld ( 5198),a
250   add a,b
260   ld (-5197),a
270   add a,b
280   ld (-5196),a
```

Essa rotina, rotulada **mrt**, não é chamada apenas no início do jogo, mas, também, após a morte de Willie, quando as variáveis são reajustadas para uma nova partida.

O MAR

Como reunimos todas as variáveis em um mesmo lugar da memória, podemos verificar o status do jogo a qualquer momento, quando precisamos corrigi-lo.

As variáveis de um byte são ajustadas pelo acumulador de oito bits, enquanto o par HL, que tem dezesseis bits, se encarrega das variáveis de dois bytes, mesmo que o valor nelas armazenado, em determinado estágio, caiba em um só byte. Isso ocorre porque os bits mais significativos da variável também devem ser ajustados.

A posição de memória −5213 contém o atraso do mar. Ela é carregada com o valor 6, para que Willie tenha



uma razoável chance de escalar a montanha antes de se afogar. Depois, poderemos modificar esse valor para acelerar o avanço da maré, tornando o jogo mais difícil e emocionante.

No início de cada tentativa de Willie, o mar deve estar bem baixo, na parte inferior do cenário. A posição do canto esquerdo da superfície do mar é armazenada em −5212 e −5211.

O número 736 é carregado nessa posição, que corresponde ao canto inferior esquerdo da tela.

A NUVEM

Na versão de *Avalanche* para o MSX, a nuvem move-se no céu acima da montanha. Os endereços −5210 e −5209, que contêm sua posição na tela, são carregados com 130, que corresponde ao ponto onde ela surge.

Mas o programa utiliza outros dados para cuidar do movimento da nuvem. Se não provocarmos um atraso, ela atravessará o céu como um jato.

Para ajustar a variável de atraso, carrega-se o endereço em que está armazenada, −52080, com o valor 3.

Precisamos também definir a direção em que a nuvem se move. Esta informação é armazenada em −5207. Se colocarmos 0 nesse endereço, ela se dirigirá para a direita; se colocarmos 1, para a esquerda. Como a rotina será inicializada tendo 0 nessa posição, a nuvem irá para a direita.

VIDA OU MORTE

O atraso do movimento da gaivota é armazenado no endereço −5206, sendo, inicialmente, ajustado com o valor 2. A posição de Willie na tela é determinada pelo conteúdo dos endereços −5205 e −5204, que são carregados com 481, valor correspondente ao lado esquerdo da base da encosta.

Uma outra variável controla a movimentação de Willie, se ele estiver de pé, correndo ou pulando. Por razões que veremos depois, ela é armazenada em dois bytes, −5302 e −5202. Como, no início do jogo, Willie está de pé, esses endereços são ajustados em 0.

A condição geral do jogo é monitorada pela variável, que fica no endereço −5201. Vamos chamá-la de variável morte. Se seu valor for 0, está tudo bem com Willie. Se for 1, nosso personagem alcançou seu prêmio e a próxima tela do jogo deverá ser chamada. Um valor 2 significa que Willie está morto! Como o jogo começa com Willie vivo, esse byte é carregado com 0.

PEDRAS E COBRAS

A variável que controla a posição da pedra é armazenada nos endereços −5200 e −5199. Essa variável é ajustada com o valor 255, que corresponde ao lado direito do topo da encosta.

As três cobras têm língua que se mexe. Como não é desejável que as línguas tenham o mesmo movimento, devemos provocar uma defasagem em seu ritmo. Para isso, carrega-se o acumulador A com o valor 0 e o registro B com a defasagem 5. O valor 0 é armazenado na variável de atraso da primeira cobra (−5198). A defasagem 5 é adicionada ao valor de A (0) pela instrução **add a,b** e o resultado, 5, vai para a variável de atraso da segunda cobra (−5197). Repete-se a operação e o resultado, 10, é armazenado na variável de atraso da terceira cobra (−5196).

MOUSE MECÂNICO E MOUSE ÓPTICO

- UM PERIFÉRICO DIFERENTE
- COMO FUNCIONA UM MOUSE MECÂNICO
- COMO FUNCIONA UM MOUSE ÓPTICO

O *mouse* é um periférico cada vez mais procurado por usuários de microcomputadores pessoais ou profissionais. Conheça-o de perto e descubra suas várias utilidades.

A busca por novas formas de interação entre o usuário e o computador tem levado ao desenvolvimento de diversos tipos de periféricos, como o joystick, a caneta óptica e o tablete de digitalização, que têm por finalidade facilitar o deslocamento de um cursor, de texto ou gráfico, sobre a tela.

Um dos periféricos dessa família é o *mouse*, ou camundongo, dispositivo que vem sendo cada vez mais usado em microcomputadores de todos os tipos.

O mouse recebeu essa denominação em virtude da semelhança física com um ratinho: é uma pequena caixa achatada, de cantos arredondados, com um a três botões planos em sua superfície dorsal e um fio (o "rabo") que sai da parte traseira, conectado ao computador.

Para usá-lo, repousa-se uma das mãos sobre a caixinha, ao mesmo tempo apoiando um ou mais dedos sobre os botões. Para efetuar algum movimento do cursor sobre a tela de vídeo, basta deslocá-lo levemente de um lado para outro sobre uma superfície dura e plana. Com um mínimo de prática, o usuário aprende a coordenar o mouse com o que acontece na tela, conseguindo grande agilidade e flexibilidade em inúmeras funções fixadas por um software especial — entre elas, selecionar itens em um menu, desenhar sobre a tela, apontar um cursor ou uma mira etc.

Os botões da parte de cima do mouse têm um papel idêntico ao do botão de disparo de um joystick, sendo usados, normalmente, para selecionar opções.

DA ORIGEM À POPULARIZAÇÃO

O mouse nasceu em um centro de pesquisas norte-americano, que estava desenvolvendo uma nova linguagem, chamada *small-talk*, e precisava de um terminal de vídeo especial para implementar os novos conceitos de interatividade que seus criadores imaginaram. O periférico que eles criaram era uma espécie de *trackball* às avessas: em vez de deslocar com a palma da mão uma esfera presa ao teclado, o usuário deslocava a esfera sobre uma superfície.

Inicialmente, o uso desse dispositivo foi bastante restrito, já que era disponível apenas para minicomputadores muito caros. Porém, a partir do aparecimento dos microcomputadores das linhas Lisa e Macintosh (fabricados pela Apple Computer nos EUA), que basearam seus sistemas operacionais na interação com o mouse, ele passou a ser fabricado também para outras linhas de microcomputadores. Em pouco tempo, tornou-se muito conhecido.

No Brasil, o mouse está disponível para microcomputadores das linhas ZX Spectrum (TK-90X e TK-95), Apple II e MSX. No exterior podem ser encontrados modelos para micros de outras linhas. Em um artigo futuro, ensinaremos algumas técnicas em BASIC para o mouse.

Embora um mouse tenha sempre a mesma aparência externa, com poucas variações de um modelo para outro, existem, na realidade, dois tipos fundamentalmente diferentes quanto ao mecanismo de funcionamento: o mouse mecânico e o mouse óptico.

O primeiro tem uma esfera de plástico ou metal maciço em sua "barriga" (ou seja, no lado voltado para a mesa). Quando o dispositivo é deslocado, a esfera gira ao redor de seu centro, acompanhando o movimento. Sensores colocados em sua superfície, dentro da caixinha, enviam ao computador os ângulos de rotação verificados em duas dimensões e um software especial, previamente carregado na memória do computador, transforma esses ângulos em coordenadas X e Y, que são usadas para posicionar o cursor na tela.

O mouse óptico funciona de acordo com um princípio totalmente diferente: um diodo luminescente (*LED*), localizado na parte inferior da caixinha, lança um feixe de luz em direção à superfície sobre a qual será deslocado o mouse. Em outro orifício, um sensor luminoso capta as variações da luz refletida pela superfície.

Ao contrário do mouse mecânico, que pode ser deslocado sobre qualquer superfície plana e não muito lisa, o mouse óptico requer uma superfície especial. Esta consiste de um tablete todo marcado com um quadriculado em cor azul ou preta. Ao se deslocar o mouse sobre ela, as linhas do quadriculado provocam uma alteração na reflexão do feixe de luz vermelha emitido pelo diodo, o que é sentido pelo software especial como uma movimentação em algum sentido. Essa informação, por sua vez, é transmitida ao computador e convertida em coordenadas X e Y.

Cada tipo de mouse apresenta vantagens e desvantagens. O modelo mecânico é mais frágil, quebrando-se facilmente, e menos sensível do que o óptico. Porém, pode ser usado sobre qualquer tipo de superfície. Já o mouse óptico requer uma superfície especial, limitada em tamanho, e apresenta ainda a desvantagem de necessitar de uma fonte de alimentação — dependendo do modelo, ela precisa ser separada da fonte de alimentação do computador.

A maioria dos periféricos de tipo mouse é conectada ao micro através da porta de joystick analógico ou de uma porta serial do tipo RS-232.

Antes de adquirir um mouse, verifique suas características para se certificar de que poderá compatibilizá-lo com seu computador.

APLICAÇÕES

O mouse é um periférico muito divertido e fácil de usar, tanto em programas para jogos de qualquer tipo, quanto em aplicações mais "sérias".

A princípio, toda aplicação que permite a utilização de canetas ópticas ou joysticks também se presta para o uso do mouse. Existem, porém, alguns programas comercialmente disponíveis, que são feitos exclusivamente para o mouse. Incluem-se, entre eles, programas de desenho sobre a tela, de seleção rápida de menus etc.

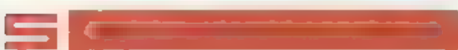
E não se esqueça: com alguns comandos simples em BASIC, não será difícil desenvolver os seus próprios programas para o irrequieto mouse.

AVALANCHE: CONTE OS PONTOS

- CONTAGEM DOS PONTOS
- TROCA DE TELA
- IMPRESSÃO DO ESCORE
- NÍVEL DE DIFICULDADE
- NÚMERO DE VIDAS

A simples obtenção de prêmios não satisfaz um aventureiro: ele precisa saber exatamente quantos pontos conseguiu com seu esforço. Monte estas rotinas e veja o placar funcionar.

A rotina que conta pontos também troca as telas, imprime o escore, o nível de dificuldade e o número de vidas.



A rotina a seguir desenha a tela apropriada, imprimindo também o escore e o número de vidas que ainda restam a Willie. Além disso, encarrega-se da execução da música.

```
 10 REM org 58676
 20 REM call lsi
 30 REM call scp
 40 REM ld hl,119
 50 REM ld a,(57343)
 60 REM ld b,48
 70 REM add a,b
 80 REM call asc
 90 REM ld a,41
100 REM call print
110 REM call tune
120 REM ret
130 REM org 58303
140 REM lsi *
150 REM org 58174
160 REM asc *
170 REM org 58217
180 REM print *
190 REM org 60006
200 REM tune *
```

O INÍCIO DO JOGO

A instrução **call lsi** chama a sub-rotina que desloca a tela para a esquerda. A tela apropriada é selecionada pela sub-rotina **elb**, que publicamos em artigo anterior.

Em seguida, a rotina **scp**, que será dada logo adiante, é chamada. Ela é responsável pela impressão do escore na tela. Não execute o programa antes de têla montado.

O par de registros HL recebe a posição de impressão do número de vidas, 119. Quando a sub-rotina **print** for chamada, HL indicará, como de costume, a posição de impressão.

O acumulador recebe o conteúdo do endereço 57343. Nessa posição de memória armazena-se o número de vidas que Willie ainda tem. O número 48 é, então, colocado em B. Para calcular o código ASCII do número a imprimir, soma-se o conteúdo desses dois registros.

Depois a rotina **asc** é chamada. Como você deve estar lembrado, essa rotina faz com que BC aponte para o padrão do algarismo a ser impresso na tabela de caracteres da ROM. A cor do número é escolhida colocando-se 41 em A. A rotina **print** é chamada para imprimir na tela o número de vidas que restam a Willie. A rotina **tune**, por sua vez, é chamada para executar a música. Essa rotina deve ser montada novamente com endereço inicial 60000 — caso contrário, o programa não funcionará.

O PLACAR

Esta é a rotina **scp** chamada pela rotina anterior

```
10 REM org 58939
20 REM scp ld hl,55
30 REM ld ix,57337
40 REM ld b,6
50 REM scq push bc
60 REM ld a,(ix+0)
70 REM ld b,48
80 REM add a,b
90 REM call asc
100 REM ld a,41
110 REM call print
120 REM inc hl
130 REM inc ix
140 REM pop bc
150 REM djnz scq
160 REM ret
170 REM org 58174
180 REM asc *
190 REM org 58217
200 REM print *
```

O par de registros HL recebe o valor 55, que corresponde à posição da tela em que será impresso o primeiro dígito do escore. O endereço do primeiro byte que contém o escore é colocado no registro-índice IX, que pode, então, ser usado como apontador.

O número de dígito do escore, 6, é colocado em B. Este contador fica temporariamente na pilha.

O byte apontado por IX é transferido em seguida para o acumulador, o que faz com que os dígitos do escore sejam colocados em A.

O número 48 é adicionado para calcular o código ASCII do algarismo e a rotina **asc** é chamada para acertar o apontador de padrões. Para a seleção da cor da letra, coloca-se 41 em A. A rotina **print** é chamada de novo para imprimir o dígito.

O par de registros HL aumenta em uma unidade para apontar a próxima posição de impressão. Observe que os dígitos mais significativos do escore são impressos primeiro. O registro IX também aumenta em uma unidade, apontando para a próxima variável do escore. Os dígitos decimais do placar são armazenados separadamente, um em cada variável, do mais significativo para o menos significativo.

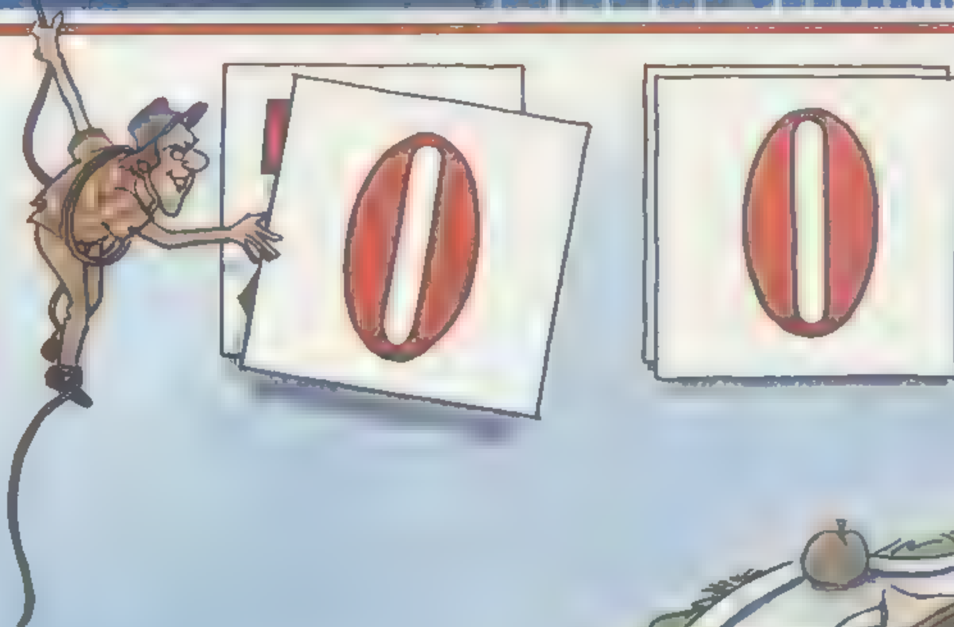
O contador é recuperado da pilha e colocado em B. A instrução **djnz** diminui seu valor em uma unidade. Se ele ainda não for zero, o processador volta ao rótulo **scq** para imprimir o dígito seguinte. Depois que o laço tiver sido executado seis vezes, imprimindo os seis dígitos do escore, o processador retornará.



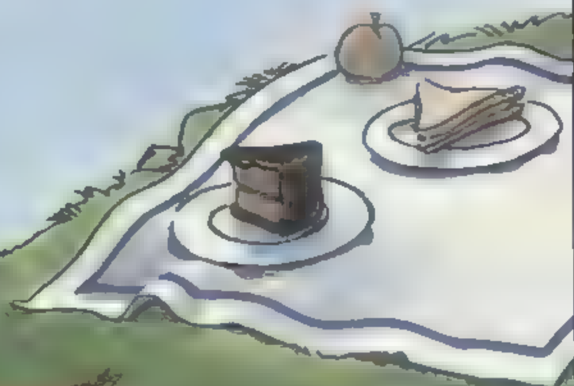
Esta rotina desloca a tela antiga e desenha a nova com o sol, o número de vidas e o placar.

```
10  ORG 19489
20  JSR $4AA5
30  LDX #1807
40  LDU #17544
50  LDB #5
60  SCPR LDA #3
```





```
70   SCPRI PULU Y
80   STY ,X++
90   DECA
100   BNE SCPRI
110   LEAX 26,X
120   DECB
130   BNE SCPR
140   JSR PRSC
150   LDX #2063
160   LDU #17574
170   LDB #5
180   SCPRZ LDA #3
190   SCPRC PULU Y
200   STY ,X++
210   DECA
220   BNE SCPRC
230   LEAX 26.X
240   DECB
250   BNE SCPRZ
260   LDA 18239
270   LDB #5
280   MUL
290   ADDD #17724
300   TFR D,U
310   LDX #2070
320   LDB #5
330   SCPRD PULU A
340   STA .X
350   LEAX 32,X
360   DECB
370   BNE SCPRD
380   JSR 30000
390   RTS
400   NOP
410   NOP
420   PRSC PSHS D X,Y
430   LDX #18240
440   LDB #6
450   LDY #1814
```





```
460  PRSCB LDA ,X
470  PSHS X,B
480  BITB #1
490  BNE ROLL
500  LDB #5
510  MUL
520  ADDD #17724
530  TFR D,X
540  PSHS Y
550  LDB #5
560  PRSCA LDA,X+
570  STA,Y
580  LEAY 32,Y
590  DECB
600  BNE PRSCA
610  PULS Y
620  LEAY 1,Y
630  ROLRET PULS B,X
640  LEAX 1,X
650  DECB
660  BNE PRSCB
670  PULS Y,X,D
680  RTS
690  ROLL LDB #5
700  MUL
710  ADDD #17724
720  TFR D,X
730  PSHS Y
740  LDB #5
750  RLLA LDA,X+
760  PSHS A
770  ANDA #15
780  LSLA
790  LSLA
800  LSLA
810  LSLA
820  ORA #15
830  STA 1,Y
840  PULS A
850  LSRA
860  LSRA
870  LSRA
880  LSRA
890  ORA #$50
900  STA ,Y
910  LEAY 32,Y
920  DECB
930  BNE RLLA
940  PULS Y
950  LEAY 2,Y
960  BRA ROLRET
```

A TELA

Inicialmente, o programa salta para a sub-rotina do endereço $4AA5, que substitui a tela antiga pela nova e desenha o sol.

Em seguida, a posição da tela em que



FIM DO JOGO!!



queremos imprimir a primeira letra da palavra "SCORE" é colocada em X. O apontador da pilha do usuário, U, recebe o endereço inicial da tabela de dados correspondentes à palavra a ser impressa. Esses dados foram colocados ali pelo programa BASIC criador de tabelas, já publicado. O registro B recebe o valor 5, pois as letras têm apenas cinco bytes de altura.

Existem cinco letras em "SCORE" mais um espaço, totalizando seis. Porém, A recebe o número 3, já que usaremos o registro Y para imprimir dois bytes de cada vez.

Os dois bytes que se encontram no topo da pilha do usuário são colocados no registro Y pela instrução **PULU Y**. A pilha do usuário é, no momento, a região da tabela de dados apontada por U. Esses dois bytes passam a ocupar as posições de memória de vídeo apontadas por X. Em seguida, o registro X tem seu conteúdo aumentado em duas unidades, apontando, então, para as duas posições seguintes.

A instrução **DECA** diminui o acumulador em uma unidade e **BNE SCPRI** repete o laço até que todos os seis bytes tenham sido impressos — o que coloca na tela apenas a primeira linha da palavra. Depois disso, a instrução **LEAX 26,X** soma 26 ao conteúdo de X, movendo-o da ponta direita da linha impressa até a posição de impressão da próxima linha da palavra. Esta fica seis bytes para a esquerda, na linha inferior. Lembre-se de que existem 32 bytes por linha e seis letras na palavra: 32 − 6 = 26.

O registro B é diminuído em uma unidade pela instrução **DECB**. O processador retorna ao rótulo **SCPR** até que todas as cinco linhas da palavra tenham sido impressas. Finalmente, a instrução **JSR PRSC** salta em direção à sub-rotina de impressão do escore.

AS CINCO VIDAS DE WILLIE

A rotina que imprime a palavra "VIDAS" é quase idêntica à que imprimiu "SCORE". Apresenta apenas duas diferenças: a posição de impressão na tela — apontada pelo registro X — e a porção da tabela de dados utilizada — apontada por U. Apesar da nova posição na tela e dos diferentes valores da tabela de dados, o processo de impressão é exatamente o mesmo.

Ao contrário da anterior, essa rotina não salta para a sub-rotina de impressão do número de vidas ao terminar, pois esta última se encontra logo a seguir na memória.

O numero de vidas que restam a Willie fica armazenado em 18239. O conteúdo dessa posição e colocado no acumulador e o número 5, em B. Em seguida, os dois registros são multiplicados.

Para imprimir o número de vidas na tela, o programa tem que obter na tabela de dados o padrão do algarismo correspondente. Cada caractere requer cinco bytes para definir seu padrão. Assim, para acharmos o endereço inicial do padrão do algarismo desejado, precisamos percorrer a tabela de dados em múltiplos de cinco.

O resultado da operação **MUL** — que multiplica os conteúdos de A e B — é colocado em D. O endereço inicial da porção da tabela que nos interessa — 17724 — é somado a esse resultado. Obtém-se, assim, o endereço inicial do padrão do algarismo correspondente ao numero de vidas.

O valor calculado é transferido para o apontador da pilha do usuário, U, de forma que essa região da tabela se transforma na pilha.

No registro X, coloca-se novamente a posição de impressão na tela; no registro B, o número de bytes necessário para escrever o algarismo, 5. O primeiro byte do padrão é retirado da pilha, colocado em A e impresso na posição de tela apontada por X.

Desta vez o registro X e acrescido de 32. Como apenas uma figura vai ser impressa, o apontador X precisa ser colocado uma posição de tela abaixo, de modo que passe a apontar para a próxima linha de pixels da figura.

B é decrementado e o processador sai do laço, executando a instrução seguinte se todos os cinco bytes já tiverem sido impressos na tela.

Para finalizar, o processador salta para a rotina 30000, que toca a música, criando o clima adequado para se iniciar a aventura. No retorno dessa sub-rotina, a instrução **RTS** devolve o controle ao BASIC.

RTS será apagada por uma outra instrução, quando o programa completo estiver montado. Ela é seguida por duas instruções **NOP** — Nenhuma **OP**eração —, que nada executam, servindo apenas para guardar o lugar dos dois últimos bytes da instrução **JSR**. Esta apagará **RTS** e acionará o laço principal que controla o programa completo.

OS NÚMEROS DO PLACAR

Os bits de cada linha de uma figura completam um byte. Se você imprimir cada um desses bytes diretamente, deixando-os muito próximos, eles terminarão por interferir uns nos outros, tornando-se, então, ilegíveis.

Para contornar essa dificuldade, a rotina que imprime os dígitos do escore controla as figuras — neste caso, números — meio byte, criando um espaço entre eles e fazendo com que os dígitos se tornem legíveis.

A rotina **PRSC** começa colocando os conteúdos dos registros D,X,Y na pilha da máquina, de modo a preservar seus valores. Mais tarde, precisaremos apenas do valor armazenado em Y. Mas nosso procedimento, aqui, vale como um lembrete: quando se programa em linguagem de máquina, convém empilhar o conteúdo de todos os registros que serão utilizados na sub-rotina. Sempre existe a possibilidade de que, posteriormente, seja necessario armazenar um importante parâmetro em um registro. Na dúvida, convém empilhá-lo.

A posição de memória 18240 armazena o byte correspondente ao primeiro dígito do escore. O valor decimal de cada um dos seis dígitos do escore — do mais significativo ao menos significativo — está guardado em seis posições de memória, de 18240 até 18245.

Como seis figuras serão impressas, carrega-se B com o número 6; Y é carregado com a posição de impressão.

O conteúdo da posição de memória apontada pelo registro X é colocado no acumulador A. Esse registro está apontado para a memória do escore. Em seguida, os valores de X e B são preservados na pilha.

A instrução **BITB #1** verifica o bit zero do registro B. Se B tem um valor par — e o bit zero, portanto, não é 1 —, a instrução **BNE ROLL** desvia o programa para a sub-rotina ROLL, que desloca todas as outras figuras meio espaço para a direita. Mas se o valor de B é impar — e, conseqüentemente, o bit zero é 1 — o processador continua com a próxima instrução.

IMPRESSÃO

Para imprimir a figura na tela, o procedimento é o mesmo utilizado no número de vidas. Multiplica-se o dígito requerido por cinco e adiciona-se o resultado ao endereço-base da tabela de dados do primeiro dígito.

Desta vez, porém, o apontador obtido é transferido para X, enquanto o apontador da posição de impressão em Y é colocado na pilha.

Carrega-se em A o byte da tabela de dados que está sendo apontado e incrementa-se o registro X. Esse byte é armazenado na posição de tela apontada por Y. O valor 32 é adicionado ao conteúdo de Y, fazendo com que este aponte para a linha de bits logo abaixo na figura. B é decrementado e o processador retorna para colocar o próximo byte abaixo, até que todos os cinco bytes que compõem a figura do dígito tenham sido impressos.

Depois disso, o apontador da posição de tela é recuperado da pilha e incrementado, passando a apontar para a próxima posição à direita.

O valor do contador em B e o apontador de memória do escore X são novamente recuperados da pilha. X é incrementado para o processamento da próxima figura à direita, enquanto B é decrementado, contando as figuras a serem impressas. Se todos os dígitos ainda não tiverem sido impressos, o processador imprime o próximo.

Neste caso, os registros Y, X e D são recuperados da pilha com os valores anteriores ao início desta rotina. Depois, o processador retorna para a posição que a chamou na rotina principal do programa.

DESLOCAMENTO DOS NÚMEROS

Se o valor do registro B for par e a rotina ROLL foi chamada, o processador localiza, na tabela de dados, o início da figura adequada ao dígito. Multiplica, então, o dígito por cinco e adiciona o resultado ao endereço-base, transferindo o valor obtido para X. Em seguida, coloca a posição de impressão na pilha e carrega o registro B com o contador de bytes.

A instrução **LDA,X+** carrega o acumulador com um dos bytes que formam a figura e incrementa o registro X para apontar o byte seguinte, que é guardado na pilha da máquina.

A instrução **ANDA #15** executa a operação lógica AND entre o conteúdo de A, uma linha de pontos da figura e o número 15, que em binário é 00001111. Essa operação equivale a colocar uma "máscara" no conteúdo de A, onde o *nybble* (grupo de quatro bits) mais significativo é apagado e o menos significativo, preservado.

Quatro instruções **LSLA** — do inglês **L**ogic **S**hift **L**eft on **A**cumulator (deslocamento lógico sobre o acumulador) — empurram o nybble menos significativo para a posição ocupada pelo nybble mais significativo, que é perdido. Essa operação é feita bit por bit.

A instrução **ORA #5** efetua a operação lógica OR entre o resultado anterior em A e o número 5, colocando amarelo — cor de fundo — no nybble menos sig-



nificativo. Após receber esse "tratamento", a linha de bits é impressa na posição apontada por Y + 1. Exibe-se, assim, na tela, a metade direita do padrão de bits, duas posições à direita da figura atual, ficando reservado o espaço onde posteriormente será colocada a metade esquerda.

O padrão de bits completo dessa linha é mais uma vez recuperado da pilha da máquina e quatro instruções **LSRA** deslocam o nybble mais significativo para a direita. A operação OR com o número hexadecimal 50 ajusta a cor de fundo desse byte para amarelo.

O byte resultante das operações acima é impresso na posição apontada por Y, ou seja, a metade esquerda do padrão de bits dessa linha é colocada na posição reservada. Com isso, as duas metades se juntam, formando a figura completa. As metades vagas desses dois bytes foram ajustadas com a cor de fundo amarela, de modo que temos, entre cada dígito do escore, um espaço de meio byte.

A instrução **LEAY 32,Y** adiciona o valor 32 a Y, para processar a linha de bits logo abaixo da figura. B é decrementado e o processador permanece no laço até que todas as cinco linhas de bits tenham sido impressas. O processador sai, então, do laço e adiciona 2 ao apontador Y.

Isso faz com que o apontador de tela se desloque duas posições para a direita, já que a figura com número ímpar ocupou duas posições adjacentes na tela — uma contendo a metade esquerda e outra, a metade direita.

Em seguida, a instrução **BRA ROLRET** retorna ao rótulo **ROLRET**, no qual os apontadores são recuperados antes da rotina voltar para processar o próximo dígito, com número ímpar.

A rotina abaixo é montada logo após a que apresentamos no artigo anterior, e faz parte do programa principal. Ela começa chamando as rotinas -11973 e -11843. A primeira desenha e desloca a montanha para a tela, e a segunda acrescenta os prêmios e os riscos de acordo com o nível de dificuldade.

O programa pode ser dividido em três partes. A primeira cria os padrões de letras e números na VRAM, usando um *buffer* de códigos ASCII e uma rotina da ROM. A segunda imprime na primeira linha da tela as palavras SCORE, VIDAS e NÍVEL, utilizando a tabela de padrões. A terceira, finalmente, imprime os seis dígitos decimais do escore, segundo o conteúdo das seis posições de memória reservadas para eles; da mesma maneira, imprime o número de vidas e o nível atual do jogo, conforme o conteúdo das posições de memória reservadas para essas variáveis.

```
10    org 53961
20    call -11973
30    call -11843
40    ld a,160
50    ld (64695),a
60    ld a,16
70    ld (64697),a
80    ld de,-14560
90    ld b,52
100    rt push bc
110    push de
120    ld a,(de)
130    call 141
140    pop de
150    inc de
160    pop bc
170    djnz rt
180    ld de,1
 90    ld a,84
200    ld b,30
210    tt push bc
220    push de
230    push af
240    ld hl,(62407)
250    add hl,de
260    call 77
270    pop af
280    inc a
290    pop de
300    inc de
310    pop bc
320    djnz tt
330    call sc
340    jp vd
350    sc ld de, 5219
360    ld (-5190),de
370    ld de,7
380    ld hl,(62407)
390    add hl,de
400    ld b,6
410    vo  push bc
420    push hl
430    ld de,(-5190)
440    ld a,(de)
450    inc de
460    ld (-5190),de
470    add a,126
480    call 77
490    pop hl
500    inc hl
510    pop bc
520    djnz vo
530    ret
540    vd ld a,(-5221)
550    add a,126
560    ld de,21
570    ld hl,(62407)
580    add hl,de
590    call 77
600    ld a,(-5228)
610    add a,126
620    ld de,30
630    ld hl,(62407)
640    add hl,de
650    call 77
660    call -12166
670    ld hl,-14054
680    call -12210
690    ret
700    end
```

ESCREVENDO NA TELA

Essa parte do programa começa na linha 40 e vai até a linha 320.

As posições de memória 64695 e 64696 da RAM contêm o endereço na VRAM do chamado *Acumulador Gráfico X*. A instrução **ld (64695),a** coloca o valor 160 nesse endereço. As posições de memória 64697 e 64698 contêm, por sua vez, o endereço do chamado *Acumulador Gráfico Y*. A instrução **ld (64697)**, a coloca o número 16 nesse endereço.

A rotina 141 da ROM imprime um caractere ASCII em qualquer posição da tela de alta resolução, ou seja, do ponto (0,0) ao ponto (255,191). Para isso, o acumulador deve conter o código ASCII do caractere e os acumuladores gráficos x e y precisam estar ajustados. O valor do acumulador x vai de 0 a 255; o do acumulador y, de 0 a 191.

Como você deve se lembrar, a tela de alta resolução é um reflexo da tabela de nomes, que contém os códigos dos padrões que aparecem no vídeo. O conjunto de bytes que formam a figura de cada padrão está na tabela de padrões da VRAM. A rotina 141 da ROM escreve o caractere ASCII na tela. Para isso, precisa criar os bytes que formam a figura desse caractere na tabela de padrões e, ainda, colocar o código do padrão na posição adequada da tabela de nomes. A rotina 141 trabalha na tela como se o modo de alta resolução acabasse de ser ligado — ou seja, com a tabela de nomes (e, consequentemente, a tela) totalmente preenchida pelos padrões de 0 a 255, dispostos seqüencialmente. Escrever na tela de alta resolução com essa rotina significa, assim, criar os bytes

que compõem a figura no padrão ou padrões correspondentes à posição apontada pelos acumuladores gráficos x e y. Essa posição equivale ao canto superior esquerdo do caractere na tela. Por exemplo, com os valores 8 e 0 nos acumuladores gráficos x e y, respectivamente, a figura seria criada no padrão de código 1; com os valores 8 e 4, nos padrões 1 e 33. Como você pode concluir, um caractere ASCII ocupa até quatro padrões.

CRIAÇÃO DOS PADRÕES

Neste programa, utilizaremos a rotina 141 para criar os padrões das 52 letras e números cujos códigos ASCII estão a partir da posição de memória -14560. O programa BASIC que apresentamos no fim do artigo encarrega-se de colocar os caracteres nessas posições. Os bytes que compõem as figuras serão colocados a partir dos endereços da tabela de padrões que correspondem ao padrão 84. Para isso, deve-se carregar o acumulador gráfico x com 160 e o acumulador gráfico y, com 16.

O par de registros DE é usado como apontador do buffer de códigos ASCII e o registro B, como contador. Em seguida, o laço **rt** coloca as figuras correspondentes aos caracteres ASCII do padrão 84 ao padrão 135. Os acumuladores gráficos x e y não precisam ser incrementados, pois, a cada chamada, a rotina 141 cuida de fazer isso.

Se você transferir a posição de chamada das rotinas -11973 e -11843 para depois do laço **rt**, poderá ver na tela a criação dos padrões.

O próximo passo consiste na impressão das palavras SCORE, VIDAS e NIVEL na primeira linha da tela. Para isso, usa-se a rotina 77 da ROM, que coloca o código do padrão na tabela de nomes da VRAM. O código deve estar, então, no acumulador A, e o endereço na tabela de nomes, no par HL. As posições 62407 e 62408 contêm o endereço inicial da tabela de nomes da VRAM. Assim, basta colocar esse endereço em HL e adicionar a ele a posição na qual queremos imprimir o padrão, de 0 a 767.

A operação acima é repetida trinta vezes pelo laço **tt**, pois, como existem vários espaços entre as palavras que estamos escrevendo, elas ocupam quase toda a primeira linha.

ROTINA DE IMPRESSÃO

A rotina que imprime os dígitos do escore é chamada pela instrução **call sc**. Em seguida, o programa salta para a rotina que imprime o dígito do escore e o dígito do número de vidas. Procedemos assim para que a rotina **sc** possa ser chamada independentemente de outras partes do programa que serão dadas mais tarde.

A rotina **sc** começa colocando o endereço -5219 nas posições de memória -5190 e -5189, através do par DE. Os valores decimais dos dígitos do escore estão armazenados em seis endereços a partir de -5219.

EXIBIÇÃO DOS DÍGITOS

A posição de impressão do primeiro dígito é colocada no par de registros HL e o contador B é ajustado em seis (número de dígitos).

O laço **vo** coloca os dígitos na tela. As posições de memória -5190 e -5189 contêm o endereço do dígito que está sendo impresso e funcionam como apontador. Para obter o padrão equivalente ao dígito no acumulador, adiciona-se seu valor decimal ao código do padrão do dígito 0, que é 126, como se pode verificar contando os padrões criados anteriormente a partir de 84.

Os dígitos decimais correspondentes ao número de vidas e ao nível do jogo estão guardados nos endereços -5221 e -5228. Como os dígitos do escore, eles são colocados pela rotina 77 da ROM nas posições 21 e 30 da tabela de nomes, respectivamente.

Depois de tratar dos dígitos do escore, do número de vidas e do nível do jogo, a rotina chama a sub-rotina -12166 — a mesma que executa a música no início do jogo.

OS CÓDIGOS ASCII

O programa BASIC apresentado a seguir coloca os caracteres ASCII correspondentes às letras e números da linha **DATA** num buffer que começa no endereço -14560. Esse programa armazena todos os caracteres na variável **R$** e utiliza a função **ASC** para obter o código correspondente a cada um deles.

```
10 CLEAR 200, 16100
20 E=-14561
30 READ R$
40 FOR N=1 TO 52
50 POKE E+N,ASC(MID$(R$,N,1))
60 NEXT N
70 DATA "SCORE 000000  VIDAS 0
NIVEL 0FIM DE JOGO 0123456789"
80 END
```





TOCANDO EM HARMONIA

- GERAÇÃO DE ACORDES
- MELODIAS SIMULTÂNEAS
- A INSTRUÇÃO **SOUND**
- TABELA DE CONVERSÃO DE NOTAS PARA O MSX

Os usuários do MSX têm o privilégio de poder utilizar acordes em suas músicas, já que esse micro possui três canais para a produção de sons. Neste artigo, você verá como harmonizar suas melodias.



Em artigos anteriores, expusemos os fundamentos da teoria musical e mostramos como transformar o micro em um instrumento, por meio de programas simples. Vimos, também, como converter partituras em dados que possam ser lidos pela máquina. Tudo isso, porém, aplicava-se apenas a melodias em que uma só nota era emitida de cada vez.

Este artigo vai um pouco mais além, apresentando programas que possibilitam a execução de acordes. Embora a teoria envolvida seja de interesse geral, só o MSX tem os recursos necessários para a execução de mais de uma nota ao mesmo tempo, em BASIC.

Nossos programas adotam as mesmas convenções dos artigos anteriores. Assim, se você quiser refrescar a memória antes de tentar assimilar as novas técnicas, reveja-os.

O QUE É UM ACORDE?

Um acorde é simplesmente um grupo de notas tocadas simultaneamente. Se pressionarmos várias teclas ao acaso no piano, obteremos um acorde. O som resultante, contudo, poderá ser bem desagradável. Para que um acorde soe bem aos nossos ouvidos, é preciso haver *harmonia*. Os acordes mais simples e harmoniosos geralmente contêm três notas e se mantêm dentro de uma escala maior — dó, ré, mi...

Se tocarmos ao mesmo tempo dó, mi (duas notas acima) e sol (mais duas notas acima) — C, E e G na escala de C maior — teremos um exemplo do mais conhecido e simples tipo de acorde, o acorde maior. O nome de um acorde é derivado da nota mais baixa que o compõe: em nosso caso, trata-se de um acorde de C maior.

Todos os acordes maiores têm quatro semitons entre a nota mais baixa e a intermediária, e três semitons entre esta última e a nota mais alta. Um segundo tipo de acorde, o acorde menor, tem três semitons entre a nota mais baixa e a intermediária, e quatro entre as duas notas superiores. Tanto os acordes maiores como os menores possuem sete semitons de comprimento, só que a nota intermediária é mais alta no acorde maior. A conseqüência, em termos de qualidade sonora, é que os maiores são geralmente mais "alegres", e os menores, mais "tristes".

Podemos construir um terceiro tipo de acorde na escala maior. Começando em si, ou B em C maior, esse acorde conterá B, mais D e F da oitava superior. Ele terá, então, dois intervalos de três semitons entre as três notas. Este é um acorde diminuto, menos usado que os dois anteriores.

A escala maior compõe-se de sete notas. Com elas podemos construir três acordes maiores, três menores e um diminuto. Os três tipos são conhecidos como tríades, já que têm três notas cada. A nota mais baixa de cada tríade é chamada *fundamental*; a nota fundamental de E menor, por exemplo, é E.

O diagrama acima mostra como esses acordes são escritos na pauta musical. Como você pode observar, as notas tocadas simultaneamente são colocadas umas sobre as outras.

COMO UTILIZAR OS ACORDES

Se uma melodia executada em C maior tem, em certo ponto, a nota C, qualquer acorde — dos tipos descritos — que também contenha a nota C poderá se harmonizar com ela, produzindo um som mais rico e agradável.

Como os acordes C e F maiores e A menor contêm a nota C, todos se harmonizarão com a melodia. A nota da música em questão pode ser qualquer uma das três notas que compõem o acorde — ou seja, toda nota se harmoniza com três diferentes tríades. C, por exemplo, é a nota mais baixa em C maior; é intermediária em A menor e é

Acordes na escala maior

a mais alta em F maior. Na realidade, se uma melodia contém a nota C, basta adicionarmos as outras duas notas para obtermos a tríade. Nesse caso, a nota da melodia original funciona não apenas como parte da melodia, mas, também, como parte do acorde.

No acorde de C maior, C não precisa ser necessariamente a nota mais baixa, podendo ter qualquer nota acima ou abaixo. Nesse caso, costuma se dizer que o acorde é invertido. A ilustração da página mostra diversos arranjos de notas — todos eles são acordes de C maior. O mesmo princípio se aplica a outros acordes.

HARMONIA AUTOMÁTICA

O programa que apresentamos a seguir obtém de linhas **DATA** as notas de uma melodia simples. Mas, ao executá-la, ele acrescenta a cada nota duas outras, mais graves, que harmonizam com ela. As duas notas são criadas por um processo aleatório; assim, a harmonia da música varia a cada nova execução. Todos os acordes, porém, constituem tríades da escala C maior.

A melodia listada ao final do programa poderá ser substituída por outra de sua própria escolha, desde que a escala seja C maior, isto é, o dó deve corresponder a O3C do comando **PLAY**. A nova melodia não deve conter nenhuma nota mais grave que O3C; caso contrário, o programa não será capaz de gerar um acorde válido.

As notas mudam ao mesmo tempo, nas três vozes e, sempre que há a alteração, uma nova harmonização é feita. O programa produz acordes de acordo com as regras, o que não significa que eles sejam sempre os mais adequados ao gosto do usuário.

A melodia *The Saints go Marching In* (*A Marcha dos Santos*) está armazenada nas linhas **DATA** do final do programa. A ilustração da página seguinte mostra duas variações da abertura da melodia que poderão eventualmente ser produzidas pelo programa. Trata-se da mesma melodia — o que muda é só a harmonia. Na primeira variação, os quatro acordes são F maior, C maior, B diminuto e G maior; na segunda, temos C maior, E menor, D menor e C maior.



```
10 R=RND(-TIME)
20 INPUT "Andamento (32-255)",T
P
30 GOSUB 3000
40 GOSUB 4000
50 GOSUB 5000
80 K1=1:K2=2:K3=3:K4=4:K5=5
90 FOR I=1 TO 32
100 READPV,T:IFPV=0THEN180
110 C=TB(PV)
120 I1%=RND(K1)*K2+K2:IFI1%=K3T
HENI2%=K5ELSEI2%=RND(K1)*K2+K4
130 I1%=C-I1%:I2%=C-I2%
140 A$="L"+STR$(T)+"N"+STR$(Q%(
PV))
150 B$="L"+STR$(T)+"N"+STR$(Q%(
TA(I1%)))
160 C$="L"+STR$(T)+"N"+STR$(Q%(
TA(I2%)))
170 PLAY A$,B$,C$
200 NEXT:END
4010 TM=23
4020 FOR I=1 TO 37
4030 Q%(I)=TM+I
4050 NEXT:RETURN
5000 DATA 1,3,5,6,8,10,12,13,15
,17,18,20,22,24,25,27,29,30,32,
34,36,37
5010 DIM TA(22):FOR I=1 TO 22:R
EAD TA(I):NEXT
5020 DATA 1,1,2,2,3,4,4,5,5,6,6
,7,8,8,9,9,10,11,11,12,12,13,13
,14
5030 DATA 15,15,16,16,17,18,18,
19,19,20,20,21,22
5040 DIM TB(37):FOR I=1 TO 37:R
EAD TB(I):NEXT:RETURN
3000 PLAY "V15","V10","V10"
3010 A$="T"+STR$(TP)
3020 PLAY A$,A$,A$
3140 RETURN
4000 DIM Q%(37)
10000 DATA 13,10,17,10,18,10
10010 DATA 20,2,13,10,17,10,18,
10
10020 DATA 20,2,13,10,17,10,18,
10
10030 DATA 20,5,17,5,13,5,17,5
10040 DATA 15,2,17,10,17,10,15,
10
10050 DATA 13,5,13,5,17,5,20,5
10060 DATA 20,10,18,2,17,10,18,
10
```

```
10070 DATA 20,5,17,5,13,5,15,5
10080 DATA 13,2
```

As linhas 10 a 80 iniciam o programa, encarregando-se de solicitar o andamento, chamar as sub-rotinas de inicialização e criar as variáveis K0 a K5, que terão valores de 0 a 5. Nas seções do programa onde a velocidade é muito importante (linhas 90-200), substituiremos os números por variáveis, já que as variáveis são manipuladas mais rapidamente que as constantes. Nessa parte do programa, todos os espaços entre os comandos devem ser removidos.

A sub-rotina da linha 3000 estabelece as características iniciais do comando **PLAY**. Note que cada canal é tratado separadamente.

A sub-rotina da linha 4000 coloca na matriz **Q%** os números das notas. O comando **PLAY** pode utilizar número no lugar de notas, o que facilita bastante o cálculo da harmonia. Uma tabela de conversão de notas em números se encontra na página 1015.

A sub-rotina da linha 5000 cria duas matrizes, **TA** e **TB**, para definir os intervalos característicos da escala C maior, possibilitando, assim, o cálculo dos acordes. **TA** contém os números das notas da escala; **TB**, a posição da nota na escala. As duas matrizes permitem que o número da nota seja obtido a partir da posição da nota na escala e vice-versa, o que é importante para o cálculo automático da harmonia.

O laço principal do programa vai da linha 90 à 200. A linha 100 lê o valor da nota e sua duração. Se o valor for 0 — pausa —, o programa salta para a linha 180. A linha 110 calcula a posição da nota na escala usando a matriz **TB**. A 120 escolhe ao acaso um dos valores 2 a 3, definindo o intervalo entre as notas da melodia e a nota intermediária. Se o intervalo for 3, a nota mais baixa deve ficar cinco notas abaixo da nota da melodia. Se for 2, a nota mais baixa pode ficar quatro ou cinco notas abaixo, o que é decidido pela linha 130.

As linhas 140 a 160 definem os cordões que servirão de operando à instrução **PLAY** da linha 170. O valor da nota original é dado por **PV**, obtido das linhas **DATA**. O valor das duas notas restantes é calculado com o auxílio da matriz **TA**, que converte a posição da nota na escala em seu valor real.

Observe que os números das notas



Arranjo para THREE BLIND MICE










nas linhas **DATA** estão codificados conforme a escala geral. A matriz **Q%** é usada para convertê-los nos números efetivamente utilizados por **PLAY**. Os intervalos são operandos do mesmo comando — utilizados junto com a letra L — e não precisam ser convertidos.

MAIS ACORDES

O programa que apresentaremos agora especifica as notas tocadas por cada um dos canais. Como será possível qualquer combinação, deixaremos para o leitor os cuidados com a harmonia.

A música fica armazenada em linhas **DATA**, como de costume. Cada uma das linhas deve conter as notas do acorde juntamente com sua duração. Note que, muitas vezes, acordes consecutivos são muito parecidos, diferindo apenas pela nota de uma das vozes.

```
10 INPUT "ANDAMENTO (1-50)";TP
20 IF TP<1 THEN 10
30 A$="T"+STR$(INT(32+223/TP)):
PLAY A$,A$,A$
40 FOR I=1 TO 48
50 READ A$,B$,C$,D
60 A$="L"+STR$(INT(1+63/D))+"O3
"+A$
70 B$="L"+STR$(INT(1+63/D))+"O3
"+B$
80 C$="L"+STR$(INT(1+63/D))+"O3
"+C$
90 PLAY A$,B$,C$
100 NEXT
1000 DATA E,O2G,O2C,6
1010 DATA D,O2B,O2G,6
1020 DATA C,O2G,O2E,12
1030 DATA E,O2G,O2C,6
1040 DATA D,O2B,O2G,6
1050 DATA C,O2G,O2E,12
1060 DATA G,D,O2B,6
1070 DATA F,C,O2A,4
1080 DATA F,C,O2A,2
1090 DATA E,O2G,O2C,12
1100 DATA G,D,O2B,6
1110 DATA F,O2B,O2G,4
1120 DATA F,O2B,O2G,2
1130 DATA E,C,O2A,10
1140 DATA G,C,O2A,2
1150 DATA O4C,E,O2G,4
1160 DATA O4C,E,O2G,2
1170 DATA B,D,O2F,2
1180 DATA A,D,O2F,2
1190 DATA B,D,O2F,2
1200 DATA O4C,E,O2G,4
1210 DATA G,E,O2G,2
1220 DATA G,F,O2B,4
1230 DATA G,F,O2B,2
1240 DATA O4C,E,O2G,2
1250 DATA O4C,E,O2G,2
1260 DATA O4C,E,O2G,2
1270 DATA B,F,O2G,2
1280 DATA A,F,O2G,2
1290 DATA B,F,O2G,2
1300 DATA O4C,E,O2G,4
1310 DATA G,E,O2G,2
1320 DATA G,D,O2F,2
1330 DATA G,D,O2F,2
1340 DATA G,D,O2F,2
1350 DATA O4C,C,O2E,4
1360 DATA O4C,C,O2E,2
1370 DATA B,D,O2F,2
1380 DATA A,D,O2F,2
1390 DATA B,D,O2F,2
1400 DATA O4C,E,O2G,2
1410 DATA G,E,O2G,2
1420 DATA G,E,O2G,2
1430 DATA G,O2B,O2F,4
1440 DATA F,O2B,O2F,2
1450 DATA E,C,O2G,6
1460 DATA D,O2B,O2G,6
1470 DATA C,O2E,O2C,12
```

As linhas 10 e 20 encarregam-se do andamento. A linha 30 inicializa a função **PLAY**. A linha 50 lê as notas do acorde e sua duração nas linhas **DATA**. Esses dados são convertidos em operandos de instrução macromusical nas linhas 60 a 80. A linha 90 emite, efetivamente, o acorde.

Ao laço **FOR...NEXT** cabe evitar que uma mensagem de erro do tipo "OUT OF DATA" interrompa a melodia.

MICRO DICAS

COMO MODIFICAR O TIMBRE

Além de produzir acordes, o MSX é capaz de modificar bastante as características do som obtido, fazendo-o ficar parecido com um instrumento específico, por exemplo.

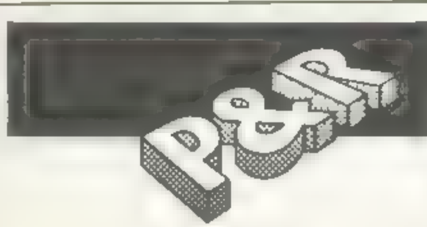
Para isso, é necessário recorrer aos registros do PSG que determinam a curva envoltória do som. Trataremos detalhadamente desse assunto em um artigo futuro.

O COMANDO SOUND

Até aqui, utilizamos apenas a função **PLAY** para executar peças musicais. Contudo, o MSX dispõe de outro comando musical, muito mais versátil: **SOUND**. As características básicas desse comando foram explicadas no artigo da página 168. Sua função é alterar o conteúdo dos registros do gerador programável de som (PSG).

Para usarmos o comando **SOUND** na produção de sons musicais, precisamos saber os valores que devem ser colocados em seus registros para produzir as notas. Para isso, consulte a tabela de conversão que está na página 1015.

O programa a seguir toca acordes de



É possível adaptar os demais microcomputadores para a produção de acordes?

Os micros que não dispõem de um processador especial de som podem produzir, entre outros efeitos sonoros interessantes, acordes semelhantes aos obtidos nos computadores da linha MSX. Para isso, é necessário combinar a frequência das diversas notas, o que requer o emprego de linguagem de máquina.

A técnica utilizada tem muita semelhança com as que foram explicadas nos artigos *Efeitos sonoros no Spectrum* (página 556) e *Apple e TK 2000 efeitos sonoros* (página 712). Embora não tenhamos abordado o assunto em INPUT, o mesmo pode ser aplicado ao micro TRS-Color.

duas notas. Os dez acordes iniciais da música *Greensleeves* estão listados nas linhas **DATA**. A melodia completa pode ser obtida no artigo da página 816.

```
10 FOR I=0 TO 10
20 READ A:SOUND I,A
30 NEXT
40 DATA 0,0,0,0,0,0,0,56,15,15
0
50 FOR I=1 TO 10
60 READ A,B,C,D,E
70 SOUND0,B:SOUND1,A
80 SOUND2,D:SOUND3,C
90 FOR J=1 TO 10*E:NEXTJ,I
100 PLAY "O5":END
110 DATA 0,253,0,253,10
120 DATA 0,213,0,253,30
130 DATA 0,189,0,213,10
140 DATA 0,169,0,213,15
150 DATA 0,159,0,213,5
160 DATA 0,169,1,28,10
170 DATA 0,189,1,28,30
180 DATA 0,225,0,225,10
190 DATA 1,28,0,225,15
200 DATA 0,253,0,225,5
```

MELODIAS SIMULTÂNEAS

O programa a seguir possibilita a execução de melodias simultâneas. Ele permite que os dados correspondentes a cada uma das vozes sejam armazenados em linhas **DATA** separadas. A melodia poderá, inclusive, ter uma velocidade diferente do acompanhamento.

Os dados para cada uma das três vozes devem terminar pelo par de valores 99,99. As outras linhas **DATA** contêm os valores das notas e suas respectivas durações. Cada linha **DATA** equivale a dois compassos. Para executar uma melodia com apenas duas vozes, basta colocar um segundo par 99,99 logo após o final dos dados da segunda voz.

O programa traz a melodia *Three Blind Mice* (*Os Três Ratinhos Cegos*) em um arranjo de três vozes mostrado no diagrama da página 1012.

```
10 R=RND(-TIME)
20 INPUT "Andamento (1-50)";TP
30 GOSUB 3000
40 GOSUB 4000
50 GOSUB 5000
80 K1=1:K2=2:K3=3
100 P1=0:P2=0:P3=0
110 GOSUB 1000
120 GOSUB 1100
130 GOSUB 1200
140 FOR I=1 TO 192
150 IF T1<>99THENT1=T1-K1:IFT1=
K0THENP1=P1+K2:GOSUB1000
160 IF T2<>99THENT2=T2-K1:IFT2=
K0THENP2=P2+K2:GOSUB1100
170 IF T3<>99THENT3=T3-K1:IFT3=
K0THENP3=P3+K2:GOSUB1200
180 FOR DL=1 TO TP:NEXT
190 NEXT
200 PLAY "O5":END
1000 T1=DA%(K1,P1+K1)
1010 PV=DA%(K1,P1):IFPV=99ORPV=
K0THENRETURN
1020 SOUND0,LQ%(PV):SOUND1,HQ%(
PV)
1030 RETURN
1100 T2=DA%(K2,P2+K1)
1110 PV=DA%(K2,P2):IFPV=99ORPV=
K0THENRETURN
1120 SOUND2,LQ%(PV):SOUND3,HQ%(
PV)
1130 RETURN
1200 T3=DA%(K3,P3+K1)
1210 PV=DA%(K3,P3):IFPV=99ORPV=
K0THENRETURN
1220 SOUND4,LQ%(PV):SOUND5,HQ%(
PV)
1230 RETURN
3000 FOR I=0 TO 10
3010 READ A:SOUND I,A
3020 NEXT
3030 DATA 0,0,0,0,0,0,0,56,15,1
0,10
3140 RETURN
4000 DIM HQ%(37),LQ%(37)
4010 TM=853:P2=2^(1/12)
4020 FOR I=1 TO 37
4030 LQ%(I)=TM-256*INT(TM/256):
HQ%(I)=TM/256
4040 TM=TM/P2
4050 NEXT:RETURN
5000 DIM DA%(3,1000)
5010 FOR VN=1 TO 3:P=0
5020 READ DA%(VN,P):READ DA%(VN
,P+1)
5030 P=P+2
5040 IF DA%(VN,P-2)=99 THEN NEX
T VN
5050 IF VN<4 THEN 5020
5060 RETURN
10000 DATA 17,6,15,6,13,12
10010 DATA 17,6,15,6,13,12
10020 DATA 20,6,18,4,18,2,17,12
10030 DATA 20,6,18,4,18,2,17,10
,20,2
10040 DATA 25,4,25,2,24,2,22,2,
24,2,25,4,20,2,20,4,20,2
10050 DATA 25,2,25,2,25,2,24,2,
22,2,24,2,25,4,20,2,20,2,20,2,2
0,2
10060 DATA 25,4,25,2,24,2,22,2,
24,2,25,2,20,2,20,2,20,4,18,2
10070 DATA 17,6,15,6,13,12
10080 DATA 99,99
20000 DATA 8,6,12,6,8,12
20010 DATA 8,6,12,6,8,12
20020 DATA 15,6,13,6,8,12
20030 DATA 15,6,12,6,13,12
20040 DATA 17,6,15,6,17,6,18,6
20050 DATA 17,6,18,6,17,6,15,6
20060 DATA 13,6,15,6,17,6,12,6
20070 DATA 13,6,12,6,5,12
20080 DATA 99,99
30000 DATA 1,6,8,6,5,12
30010 DATA 1,6,8,6,5,12
30020 DATA 12,6,10,6,1,12
30030 DATA 12,6,8,6,10,12
30040 DATA 8,6,6,6,8,6,12,6
30050 DATA 8,6,8,6,8,6,6,6
30060 DATA 5,6,6,6,8,6,6,6
30070 DATA 8,6,8,6,1,12
30080 DATA 99,99
```

As linhas 10 a 80 dão início ao programa. Encarregam-se de definir o andamento, chamar várias sub-rotinas e determinar as variáveis **K1**, **K2** e **K3**, que substituem os números 1, 2 e 3 onde a velocidade for crítica.

A sub-rotina 3000 acerta a configuração inicial do PSG, ativando-o para a produção de notas musicais nos três canais disponíveis. A primeira voz terá volume mais elevado que as demais.

A sub-rotina 4000 coloca nas matrizes **LQ%** e **HQ%** os valores dos bytes menos significativo e mais significativo, correspondentes às frequências das notas musicais. Isso permite que as notas sejam definidas pelo seu número na escala geral, e não pelos complicados valores dos registros do PSG.

A sub-rotina 5000 lê os valores e durações das notas em linhas **DATA**, colocando-os na matriz **DA% (3,1000)**. O primeiro índice corresponde ao número da voz, o segundo, à posição da nota na melodia. Esta será lida e depois executada, ao contrário dos programas apresentados anteriormente. **P1**, **P2** e **P3** funcionam como apontadores de cada uma das vozes.

A sub-rotina das linhas 1000 a 1030 toca uma nota no canal A; **P1** aponta o par de dados corrente que especifica nota e duração; **T1** é o contador que controla a duração da nota.

A linha 1010 coloca em **PV** o valor da nota. Se esse valor for 99, o programa chegou ao fim dos dados referentes ao canal A; se for zero, trata-se de uma pausa. Em ambos os casos, a sub-rotina termina numa instrução **RETURN**. Nas demais situações, a linha 1020 coloca os bytes calculados com o auxílio de **HQ%** e **LQ%** nos registros adequados do gerador programável de som, usando o comando **SOUND**. As sub-rotinas 1100 e 1200 executam as mesmas operações nos canais B e C.

A linha 100 acerta os ponteiros para começar a leitura de **DA%**. As linhas 110 a 130 chamam as sub-rotinas 1000, 1100 e 1200 pela primeira vez.

LAÇO PRINCIPAL

O **FOR...NEXT** entre a linha 140 e a linha 200 constitui o laço principal do programa. A linha 150 testa o contador **T1**, que contém o valor da duração da nota corrente no canal A: um valor 99 indica o fim dos dados. Se **T1** tiver outro valor, é diminuído em uma unidade. Quando chegar a zero, a execução da nota terá terminado e outra será obtida pela sub-rotina 1000. Enquanto o valor zero não for atingido, a nota continua soando.

As linhas 150 e 160 executam a mesma operação nas duas outras vozes. A linha 170 provoca um atraso proporcional ao andamento. Na linha 200, a instrução **PLAY "05"** é utilizada para interromper a música.

Observe que, quando utilizamos o comando **PLAY**, as durações eram definidas por durações de nota. Já com o **SOUND**, os valores são proporcionais à duração desejada para cada nota.

Para terminar, apresentamos as modificações que precisam ser feitas no programa que calcula a harmonia automática de acordes para que funcione com o comando **SOUND**:

```
20 INPUT "Andamento (1-50)";TP
140 SOUND0,LQ%(PV):SOUND1,HQ%(P
V)
150 SOUND2,LQ%(TA(I1%)):SOUND3,
HQ%(TA(I1%))
160 SOUND4,LQ%(TA(I2%)):SOUND5,
HQ%(TA(I2%))
170 FOR DL=1 TO (51-TP)*40/T:NE
XT
200 NEXT:PLAY "O5":END
3000 FOR I=0 TO 10
3010 READ A:SOUND I,A
3020 NEXT
3030 DATA 0,0,0,0,0,0,0,56,15,1
0,10
4000 DIM HQ%(37),LQ%(37)
4010 TM=853:P2=2^(1/12)
4020 FOR I=1 TO 37
4030 LQ%(I)=TM-256*INT(TM/256):
HQ%(I)=TM/256
4040 TM=TM/P2
```

TABELA DE CONVERSÃO PARA O MSX

Escala geral	Byte mais significativo	Byte menos significativo	Número	Nota	Escala geral	Byte mais significativo	Byte menos significativo	Número	Nota
1	3	85	25	O3C	20	1	28	44	G
2	3	37	26	C+	21	1	12	45	G+
3	2	247	27	D	22	0	253	46	A
4	2	205	28	D+	23	0	239	47	A+
5	2	165	29	E	24	0	225	48	B
6	2	127	30	F	25	0	213	49	O5C
7	2	91	31	F+	26	0	201	50	C+
[illegible]	2	57	32	G	27	0	189	51	D
9	2	25	33	G+	28	0	179	52	D+
10	1	251	34	A	29	0	169	53	E
11	1	222	35	A+	30	0	159	54	F
12	1	195	36	B	31	[illegible]	150	55	F+
13	1	170	37	O4C	32	0	142	56	G
14	1	146	38	C+	33	0	134	57	G+
15	1	123	39	D	34	0	126	58	A
16	1	102	40	D+	35	0	119	59	A+
17	1	82	41	E	36	0	112	60	B
18	1	63	42	F	37	0	106	61	O6C
19	1	45	43	F+					

JOGOS DE GUERRA: PRIMEIROS PASSOS

A criação de jogos de guerra no computador é uma atividade simplesmente fascinante. Aprenda as técnicas envolvidas e depois mobilize seu micro para esta batalha de INPUT.

Embora fossem conhecidos há milhares de anos, os jogos de guerra (*Wargames*) estavam restritos, até há bem pouco tempo, aos quartéis-generais e escolas militares, devido à exigência de tabuleiros imensos e à grande quantidade de peças necessárias para representar as forças em conflito. O advento dos computadores permitiu que se popularizassem, já que a máquina confina toda a massa de dados em sua memória, utiliza a tela gráfica como mapa e pode até fazer o papel de jogador.

Esses jogos desenvolveram-se originalmente como elemento de apoio ao ensino de estratégia militar. Hoje, sua reprodução no computador tem seduzido uma multidão de usuários. E, embora o objetivo principal de um jogo de guerra seja sempre a eliminação do inimigo, ele se relaciona muito mais, quanto à diversão proporcionada, ao xadrez do que a um videogame de ação, cheio de tiros e pistolas laser.

Durante um jogo de guerra, a tela do micro exibe um mapa da região de batalha, indicando, em geral, as posições dos dois inimigos. O computador possibilita uma simulação da realidade muito mais fiel que a proporcionada pelos jogos tradicionais, na medida em que nos dá a chance de posicionar nossas forças sem que o inimigo as veja. As unidades só se tornam visíveis quando efetivamente descobertas.

Porém os jogos que os estrategistas utilizam costumam envolver os mínimos detalhes de uma guerra — o que não temos condições de fazer em um computador doméstico.

Alguns jogos de computador são destinados exclusivamente a adversários humanos. O programa que apresentamos em INPUT, porém, permite que você jogue contra seu micro.

PLANEJAMENTO

Uma guerra consiste em mover "unidades de combate" (em geral, soldados, mas, também, tanques ou canhões) até a posição do inimigo, para tentar submetê-lo ou destruí-lo.

Os ingredientes básicos de um jogo de guerra são as duas forças adversárias, seus movimentos e o combate entre as unidades. Com isso em mente, já podemos planejar o jogo. A batalha será na terra, no mar ou no ar? Queremos um jogo estratégico em larga escala, com muitos exércitos envolvidos; um jogo tático entre dois exércitos em um único campo de batalha; ou só escaramuças entre combatentes individuais?

O período da História em que se dá a guerra também é importante, pois determina o tipo de tecnologia bélica e as características dos combatentes. Podemos recriar batalhas famosas, como as travadas por Napoleão em território russo, ou inventar nossa própria guerra, dando asas à imaginação.

O passo seguinte consiste na definição das regras do jogo. Elas determinarão as circunstâncias que ajudarão ou prejudicarão cada um dos oponentes, estendendo-se aos menores detalhes incluídos no jogo. Você pode querer, por exemplo, que os soldados tenham a opção de usar armaduras. Isso os protegerá durante a luta, mas, em compensação, tornará mais lento o movimento de avanço ou de retirada.

É muito tentador enriquecer o jogo com mil detalhes, aproximando-o ao máximo da realidade. Porém o tamanho da memória limita a quantidade de detalhes e a complexidade das regras. Convém, assim, manter a atenção sobre os pontos principais:

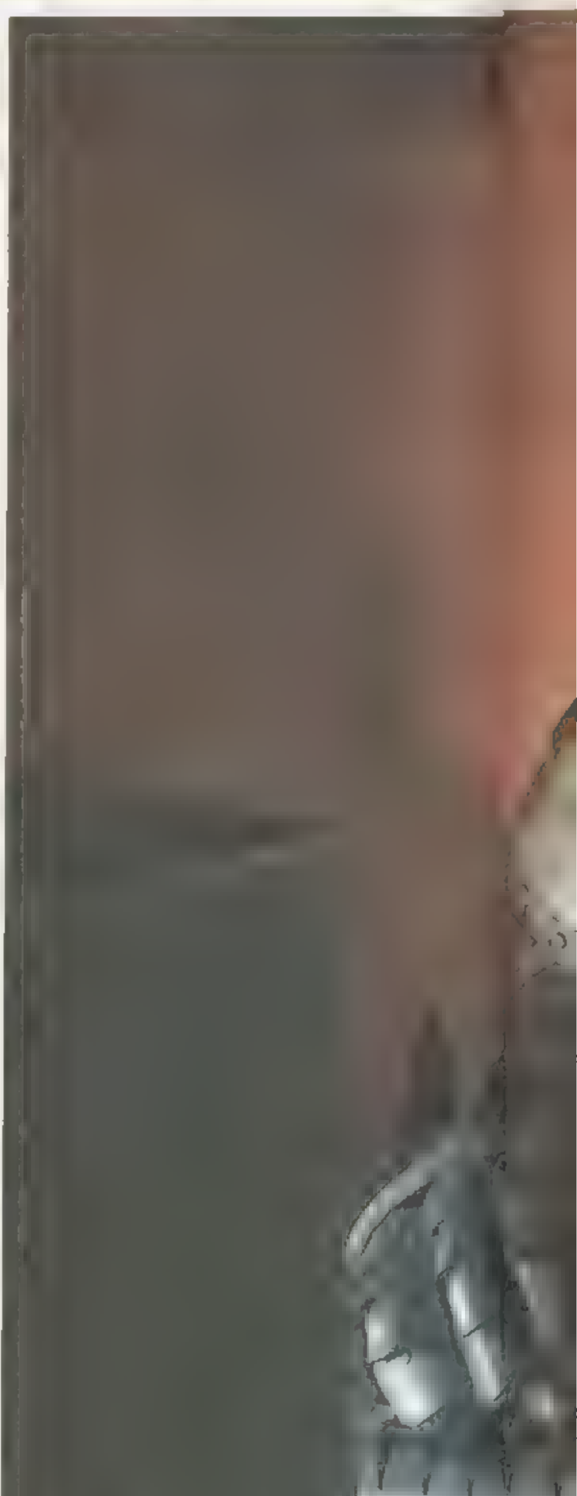
- Local: informações, na forma de mapas, a respeito do local onde se encontram as tropas e do tipo de terreno da região — como afeta o movimento e que tipo de proteção oferece.

- As tropas: quantos homens são; que tipo de munição e proteção usam; a que velocidade se locomovem.

- Movimento: que distância uma unidade poderá se mover; como o tipo de terreno afetará o movimento.

- Comandos: definir como serão dadas as ordens e se as tropas poderão desobedecê-las ou não.

- O computador como inimigo: definir se a máquina será mais ou menos inteli-



- JOGOS DE GUERRA NO TABULEIRO E NO COMPUTADOR
- ORGANIZAÇÃO DO JOGO
- O CENÁRIO
- A ÉPOCA
- REGRAS DA BATALHA
- FUNCIONAMENTO DA TELA
- BLOCOS GRÁFICOS
- LIMPEZA DA ÁREA DE TEXTOS



gente, variando, assim, o nível de dificuldade do jogo.

• Combate: tipo de combate — balístico ou corpo a corpo; tipo de projetil — de simples flechas até mísseis intercontinentais.

NOSSO JOGO

Uma vez escolhidos o tipo e os componentes da guerra e o período histórico em que ela ocorre, precisamos determinar como tudo isso será representado no micro. Devemos considerar dois aspectos: o ponto de vista do jogador e o do computador, e como o jogo será apresentado a cada um deles.

Nesta série de cinco artigos tentaremos mostrar as técnicas básicas de programação criando em seu micro um jogo que denominaremos *Capa e Espada*. Ele consiste em uma batalha tática entre dois exércitos do período medieval. Essa guerra, contudo, não se passa em nenhuma data definida.

Como a maioria dos jogos de guerra, *Capa e Espada* mostra na tela um mapa, com a disposição das tropas e os acidentes geográficos da região. Os jogadores (no caso, você e o computador) agem como comandantes das unidades, devendo tomar as decisões estratégicas bem como dar as ordens relativas ao comportamento da tropa.

Instruções mais detalhadas serão fornecidas no quarto artigo desta série, quando o programa estiver completo. De um modo geral, cada jogador pode escolher entre dar ordens ou deixar as tropas como estão. O jogo se desenrola com os jogadores movimentando seus comandados pelo campo de batalha, alternadamente, a fim de organizar a disposição das forças. Esse movimento pode ou não resultar em conflito. O efeito dos eventuais choques entre as unidades é determinado pelo tipo e pelo poder das tropas envolvidas — além de uma pitadinha de sorte. O jogo continua até que um dos exércitos tenha sido praticamente destruído.

O programa mostrará as informações gráficas relevantes — o mapa com os exércitos — continuamente, reservando uma porção da tela para textos.

Usaremos blocos gráficos para criar o mapa, de maneira que instruções do tipo **PRINT** possam controlar tanto o mapa como a área de textos. No caso do TRS-Color, os blocos gráficos serão definidos por instruções **DRAW** na tela de alta resolução, como temos feito na maioria dos programas de INPUT. Utilizaremos também uma rotina para im-





primir textos na tela gráfica. No Apple e no TK-2000, instruções **POKE** controlarão o mapa, enquanto o texto ficará nas quatro linhas inferiores.

Neste jogo há quatro tipos de terreno: campo aberto, vilas, florestas e montanhas. Empregaremos espaços em branco para representar o campo — portanto, não será preciso um bloco gráfico especial. Usaremos dois tipos de bloco para representar as montanhas e um tipo para cada terreno restante.

Os exércitos terão oito unidades de combate: o líder e seus cavaleiros, os nobres vassalos (sargentos), duas unidades de lanceiros, duas de arqueiros e duas de camponeses. As duas primeiras unidades, formadas por nobres, requerem blocos gráficos diferentes. Para cada par de unidades restantes será usado um mesmo caractere duas vezes.

Teremos, assim, nove blocos gráficos: número um, vila; dois, florestas; três e quatro, montanha; cinco, o líder (representado por uma bandeira); seis, sargentos (representados por uma maça — arma medieval composta por um cabo com corrente e uma bola de ferro na ponta); sete, lanceiros (um escudo); oito, arqueiros (arco e flecha) e nove, camponeses (espada).

OS BLOCOS GRÁFICOS

Os programas listados a seguir criam os blocos gráficos descritos acima.

S

```
 210 FOR k=1 TO 9
 220 READ a$
 230 FOR I=0 TO 7
 240 READ a
 260 POKE USR a$+I,a
 270 NEXT I
 290 NEXT K
2540 REM Limpa tela de texto
2550 FOR k=17 TO 21: PRINT AT k
,0;"
     "  NEXT k
2570 DATA "a",16,16,60,126,255,
189,231,231
2590 DATA "b",16,56,84,16,56,84
,146,16
2610 DATA "c",8,20,34,65,6,8,16
,224
2630 DATA "d",0,48,72,132,2,0,0
,0
2650 DATA "e",128,240,255,252,1
43,128,178,128
2670 DATA "f",64,240,72,68,68,6
8,78,68
2690 DATA "g",255,231,231,129,1
29,231,102,60
2710 DATA "h",249,70,38,25,9,5,
3,1
2730 DATA "i",1,2,4,8,16,160,64
,160
```



```
200 SCREEN 1,2:KEY OFF:COLOR 1,
15,15
210 FOR I=0 TO 71
220 READ A
225 VPOKE BASE(7)+192*8+I,A
230 VPOKE BASE(7)+208*8+I,A
235 VPOKE BASE(7)+224*8+I,A
240 VPOKE BASE(7)+240*8+I,A
245 NEXT
250 FOR I=0 TO 1
260 VPOKE BASE(6)+24+I,4*16+15
265 VPOKE BASE(6)+26+I,6*16+15
270 VPOKE BASE(6)+28+I,12*16+15
280 NEXT
290 VPOKE BASE(6)+30,13*16+15
2570 DATA 16,16,60,126,255,189,
231,231
2590 DATA 16,56,84,16,56,84,146
,16
2610 DATA 8,20,34,65,6,8,16,224
2630 DATA 0,48,72,132,2,0,0,0
2650 DATA 128,240,255,252,143,1
28,128,128
2670 DATA 64,240,72,68,68,68,78
,68
2690 DATA 255,231,231,129,129,2
31,102,60
2710 DATA 249,70,38,25,9,5,3,1
2730 DATA 1,2,4,8,16,160,64,160
```



```
210 EE = 768:T = 16384: FOR I =
 EE TO EE + 30 * 8 - 1
220  READ A: POKE I,A
230  NEXT
2570  DATA  64,64,80,84,85,17,2
1,21
2575  DATA  0,0,2,10,42,34,42,4
2
2580  DATA  0,32,40,8,32,40,10,
2
2585  DATA  1,5,21,17,5,21,81,6
5
2590  DATA  64,48,12,3,0,0,112,
15
2595  DATA  1,6,24,96,120,7,0,0
2600  DATA  64,48,12,3,0,0,0,0
2605  DATA  1,6,24,96,0,0,0,0
2610  DATA  129,149,213,213,193
,129,129,129
2615  DATA  128,128,138,130,138
,128,128,128
2620  DATA  132,149,196,132,132
,132,196,132
2625  DATA  128,128,128,130,130
,130,138,130
2630  DATA  213,149,149,129,149
,148,208,192
2635  DATA  170,170,170,160,170
,138,130,128
2640  DATA 192,192,192,208,196,
193,213,128
2645  DATA 128,128,128,130,136,
160,170,128
2650  DATA  128,128,128,196,212
,148,149,145
2655  DATA  160,168,138,130,128
,128,128,128
2660  DATA  128,128,168,160,168
,128,128,128
2665  DATA  192,212,213,213,193
,192,192,192
2670  DATA  128,128,128,160,160
,160,168,160
2675  DATA  144,212,145,144,144
,144,145,144
2680  DATA  170,170,170,130,170
,168,160,128
2685  DATA  213,212,212,192,212
,148,133,129
2690  DATA  128,128,128,160,136
,130,170,128
2695  DATA   129,129,129,133,14
5,193,213,128
2700  DATA  130,138,168,160,128
,128,128,128
2705  DATA  128,128,128,145,149
,148,212,196
2710  DATA  0,0,0,0,0,0,0,0
2715  DATA  0,0,0,0,0,0,0,0
```



```
210 FOR K=1 TO 9
220 READ A$
230 UCS(K)=A$
290 NEXT K
2570 DATA BR2D2L2D5RU2R3D2R3U4L
2UL2D2
2590 DATA BR2R3DRDLDL2ND4L3U2R3
DL3
2610 DATA BD3E2R2UD2RFGLG3
2630 DATA BD3E2RF3
2650 DATA ND7FR2FD5UNR3L2
2670 DATA ND7FR4DNR2DNFL3UR2
2690 DATA R5ND6DL5D5R3DL
2710 DATA NR3DRF6U4LHE2DL
2730 DATA BR7G7E2UNF3H2DR
```

Todos os programas incluem várias linhas **DATA**, utilizando-as de uma forma diferente para a criação dos blocos gráficos. O Spectrum coloca seus blocos nos caracteres 124 a 133. O MSX usa a tela de texto de 32 colunas. Não recorreremos à tela gráfica porque não vamos empregar comandos de alta resolução, e é mais fácil escrever na tela de textos.

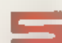
As linhas que vão de 210 a 245 desenham os blocos gráficos na tabela de padrões. Cabe às linhas 250 a 280 colori-los. São feitas quatro cópias dos nove blocos, uma de cada cor. Embora os blocos de terreno e os dois exércitos tenham cada qual uma só cor é mais fácil criar todos os blocos com as quatro cores do que definir as cores de blocos individuais.

O USO DO POKE

No Apple e no TK-2000, montamos os blocos gráficos com **POKE**, em vez de usar o comando **DRAW**. Fizemos isso para economizar memória, pois seriam necessários muitos bytes para definir uma tabela com tantas figuras. O banco de blocos ficará armazenado em uma região normalmente não utilizada da página três — assim, não ocupará espaço disponível ao BASIC. Na realidade, serão criados dezoito blocos, dois para cada unidade ou terreno, já que, para obter cor, só usamos colunas pares ou ímpares. Para maiores detalhes sobre esse tipo de bloco gráfico, veja os artigos das páginas 489 e 507.

A ÁREA DE TEXTOS

As linhas que se seguem são utilizadas para a impressão de mensagens no vídeo do micro.



```
2540 REM   Limpa tela de texto
2550 FOR k=17 TO 21: PRINT AT k
,0,"
     ": NEXT k
2555 RETURN
```



```
2540 REM Limpa área de texto
2550 FOR I=576 TO 767
2555 VPOKE BASE(5)+I,32
2560 NEXT: RETURN
```



```
2540  REM  LIMPA
2545  HOME : GOSUB 30000
2550  RETURN
```



```
2540 REM  LIMPA JANELA DE TEXTO
2550 PMODE 0,4:PCLS: PMODE 3,1
2560 RETURN
```

O programa trata a tela como duas "janelas" — uma de textos, ocupando uma pequena área do vídeo, e outra gráfica, contendo o mapa da região e a disposição das tropas adversárias.

No desenvolvimento do jogo, a janela de textos precisará ser limpa e reescrita com freqüência. A janela gráfica, porém, é mais constante, exigindo apenas pequenas alterações nas posições das unidades.

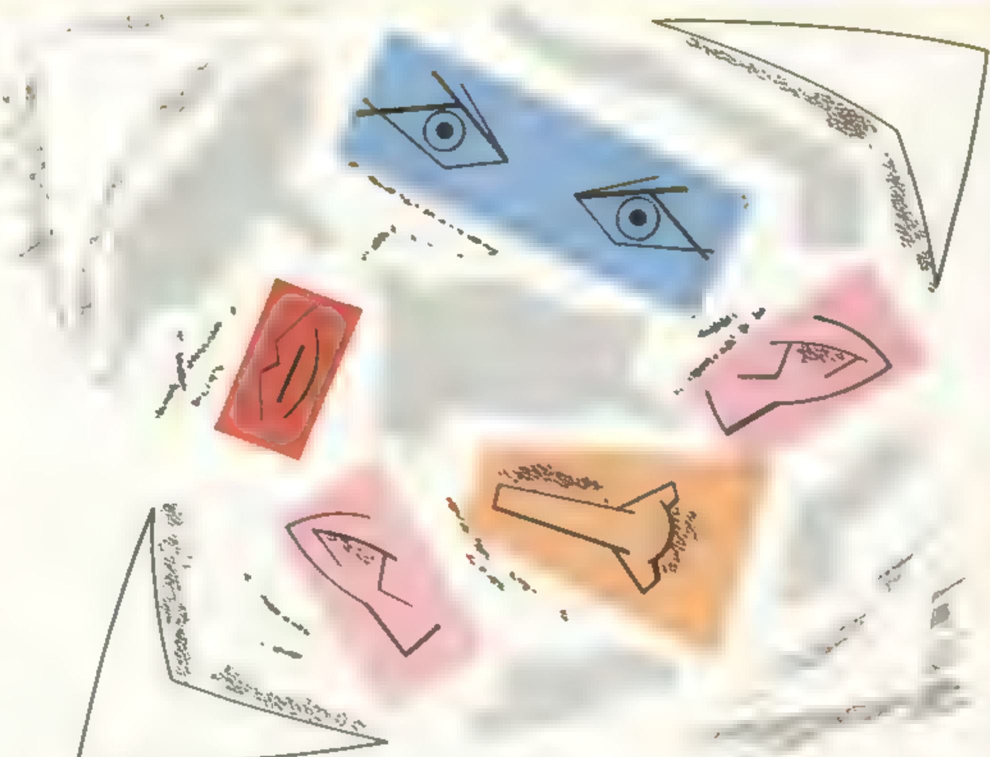
Todos os microcomputadores, com exceção do Apple e do TK 2000, tratam a tela como uma unidade, de maneira que essas rotinas se encarregam de apagar exclusivamente a área de textos, deixando o mapa intacto.

No próximo artigo da série que aqui iniciamos veremos como desenhar o mapa da batalha e movimentar as unidades que se confrontam.

MONTAGEM DE DESENHOS

- PRINCÍPIOS BÁSICOS DO PAC
- LIGANDO PONTOS
- MOVIMENTAÇÃO DA FIGURA
- EXPANSÃO, CONTRAÇÃO E ROTAÇÃO

A construção de imagens gráficas no computador é, quase sempre, um processo difícil. O Projeto Assistido pelo Computador (PAC) poderá simplificar bastante o seu trabalho.



Existem sistemas destinados a facilitar a elaboração de desenhos cujos componentes se repetem muitas vezes. O projeto de um supermercado, por exemplo, exige que se distribua, da melhor forma possível, uma série de objetos idênticos — prateleiras, refrigeradores etc. — em uma certa área. Esse tipo de sistema, denominado Projeto Assistido pelo Computador (PAC), pode se apresentar com vários níveis de sofisticação. Mas, qualquer que seja seu grau de complexidade, o PAC baseia-se em duas operações básicas: traçar retas e remodelar figuras prefixadas.

Este artigo irá ajudá-lo a construir duas versões simples de um PAC.

TRAÇANDO RETAS

A técnica que examinaremos agora reproduz a maneira mais simples de se executar um desenho com o auxílio de um lápis e uma régua.

Primeiro, fazemos o traço inicial. Depois, a partir de uma das extremidades desse traço, desenhamos outra reta e assim por diante, sempre ligando a ponta do último segmento a um ponto imaginário. No computador, fixamos um ponto na tela e levamos o cursor até a posição onde um segundo ponto será fixado. Este irá determinar a reta que deve ser traçada.

Para facilitar ainda mais o processo, o computador traça uma reta a cada nova posição definida pelo cursor, apagando a anterior. Ao chegar à posição definitiva, o usuário digita algum tipo de código destinado a impedir que o computador apague a última linha, deixando-a fixa na tela. Em seguida, pode-se movimentar o cursor para outro ponto qualquer.

Experimente fazer desenhos com o programa que se segue.

S

```
10 BORDER 0: PAPER 0: INK 7:
CLS
15 LET inicio=80: LET estica=
130: LET desenha=210: LET fix
a=240
17 DRAW 0,175: DRAW 255,0:
DRAW 0,-175: DRAW -255,0
20 GOSUB inicio
40 GOSUB estica
50 IF INKEY$<>"q" THEN  GOTO
 40
60 STOP
90 OVER 1
100 LET x=128: LET y=86
115 PLOT x,y
120 RETURN
140 IF INKEY$=" " THEN  GOSUB
fixa: GOTO 150
145 GOSUB desenha
150 IF INKEY$="z" THEN  LET x=
x-1
160 IF INKEY$="x" THEN  LET x=
x+1
170 IF INKEY$="k" THEN  LET y=
y+1
180 IF INKEY$="m" THEN  LET y=
y-1
190 GOSUB desenha
200 RETURN
220 PLOT x,y: DRAW x-PEEK
23677,y-PEEK 23678
230 RETURN
250 OVER 0: GOSUB desenha
255 OVER 1
280 RETURN
```



```
10 PCLEAR 8:PMODE 4,5:PCLS PMOD
E 4,1:PCLS:SCREEN 1,1
20 V=247:CX=127:CY=95:X=127:Y=9
5
30 GOSUB 100
40 FOR K=1 TO 4:PCOPY K+4 TO K:
NEXT
50 IF PEEK(338)<>191 THEN 30
60 CLS:END
100 IF PEEK(345)=V GOSUB 300
110 IF PEEK(341)=V AND Y>0 THEN
 Y=Y-1
120 IF PEEK(342)=V AND Y<191 TH
EN Y=Y+1
```

```
130 IF PEEK(343)=V AND X>0 THEN X=X 1
140 IF PEEK(344)=V AND X<255 THEN X=X+1
200 LINE(CX,CY)-(X,Y),PSET
210 RETURN
300 GOSUB 200
310 FOR K=1 TO 4:PCOPY K TO K+4:NEXT
320 CX=X:CY=Y
330 RETURN
```



```
10  HGR2
20 CX = 140:CY = 96:X = CX:Y = CY
100  GET A$
102  IF A$ = " " THEN CX = X:CY = Y: GOTO 110
103  HCOLOR= 0: HPLOT CX,CY TO X,Y
110  IF A$ = "Z" AND X > 1 THEN X = X - 2
120  IF A$ = "X" AND X < 278 THEN X = X + 2
130  IF A$ = "." AND Y < 190 THEN Y = Y + 2
140  IF A$ = ";" AND Y > 1 THEN Y = Y - 2
155  IF A$ = "S" THEN  STOP
160  HCOLOR= 3: HPLOT CX,CY TO X,Y
170  GOTO 100
```



```
10  HGR2
20 CX = 140:CY = 96:X = CX:Y = CY
100  GET A$
102  IF A$ = " " THEN CX = X:CY = Y: GOTO 110
103  HCOLOR= 0: HPLOT CX,CY TO X,Y
110  IF A$ =  CHR$ (8) AND X > 1 THEN X = X - 2
120  IF A$ =  CHR$ (21) AND X < 278 THEN X = X + 2
130  IF A$ =  CHR$ (113) AND Y < 190 THEN Y = Y + 2
140  IF A$ =  CHR$ (112) AND Y > 1 THEN Y = Y - 2
155  IF A$ = "S" THEN  STOP
160  HCOLOR= 3: HPLOT CX,CY TO X,Y
170  GOTO 100
```

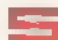


```
10 SCREEN 2
20 CX=128:CY=96:X=CX:Y=CY
100 A$=INKEY$:IF A$="" THEN 100
103 IF A$<>" " THEN LINE (CX,CY)-(X,Y),4 ELSE CX=X:CY=Y
110 IF ASC(A$)=29 AND X>2 THEN X=X-2
120 IF ASC(A$)=28 AND X<255 THEN X=X+2
130 IF ASC(A$)=31 AND Y<191 THEN Y=Y+2
140 IF ASC(A$)=30 AND Y>1 THEN Y=Y-2
160 LINE (CX,CY)-(X,Y),11
170 GOTO 100
```

O funcionamento do programa é bem simples. Ele fixa um ponto no centro da tela e outro em uma posição determinada por quatro teclas — **Z** (esquerda), **X** (direita), **K** (acima) e **M** (abaixo). O APPLE usa **;** (acima), **.** (abaixo) e **S** (para interromper o programa). Os microcomputadores das linhas MSX e o TK-2000 utilizam as teclas de controle do cursor (flechinhas).

Uma reta é traçada continuamente entre os dois pontos assim determinados. Quando alcançar sua posição definitiva, ela será fixada na tela, bastando, para isso, que se pressione a barra de espaço. Você poderá, então, prosseguir em outra direção, usando as mesmas quatro teclas.

Digite agora estas linhas adicionais para conseguir o efeito de tirar o lápis do papel.



```
15 LET inicio=80: LET estica=130: LET desenha=210: LET fixa=240: LET apaga=210
17 GOSUB apaga
144 IF INKEY$="d" THEN  GOSUB apaga: GOTO 150
310 CLS
320 DRAW 0,175: DRAW 255,0: DRAW 0,-175: DRAW -255,0
330 GOSUB inicio
340 RETURN
```



```
105 IF PEEK(339)=191 THEN PMODE 4,5:PCLS:PMODE 4,1
301 A$=INKEY$
302 A$=INKEY$:IF A$="" THEN 302
304 IF A$="D" THEN 320
```



```
150  IF A$ = "D" THEN CX = X:CY = Y
```



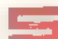
```
105  IF A$ = "D" THEN CX = X:CY = Y
```

Agora, ao pressionar a tecla **D**, a última linha traçada será apagada. O ponto onde o cursor se encontra será definido como inicial.

MONTAGEM DE UM DESENHO

Alguns tipos de PAC apresentam em um menu várias figuras previamente definidas. Se o problema fosse a decoração de uma casa, por exemplo, o menu incluiria os desenhos de móveis, esquadrias e outros elementos, que seriam transportados para a tela e ajustados à vontade ao projeto do decorador. Para tais ajustes, é possível recorrer a ampliações, reduções e rotações.

O programa a seguir oferece, em seu menu, cinco figuras geométricas. Elas podem ser fixadas na tela e manipuladas de acordo com as instruções acrescentadas após a listagem.

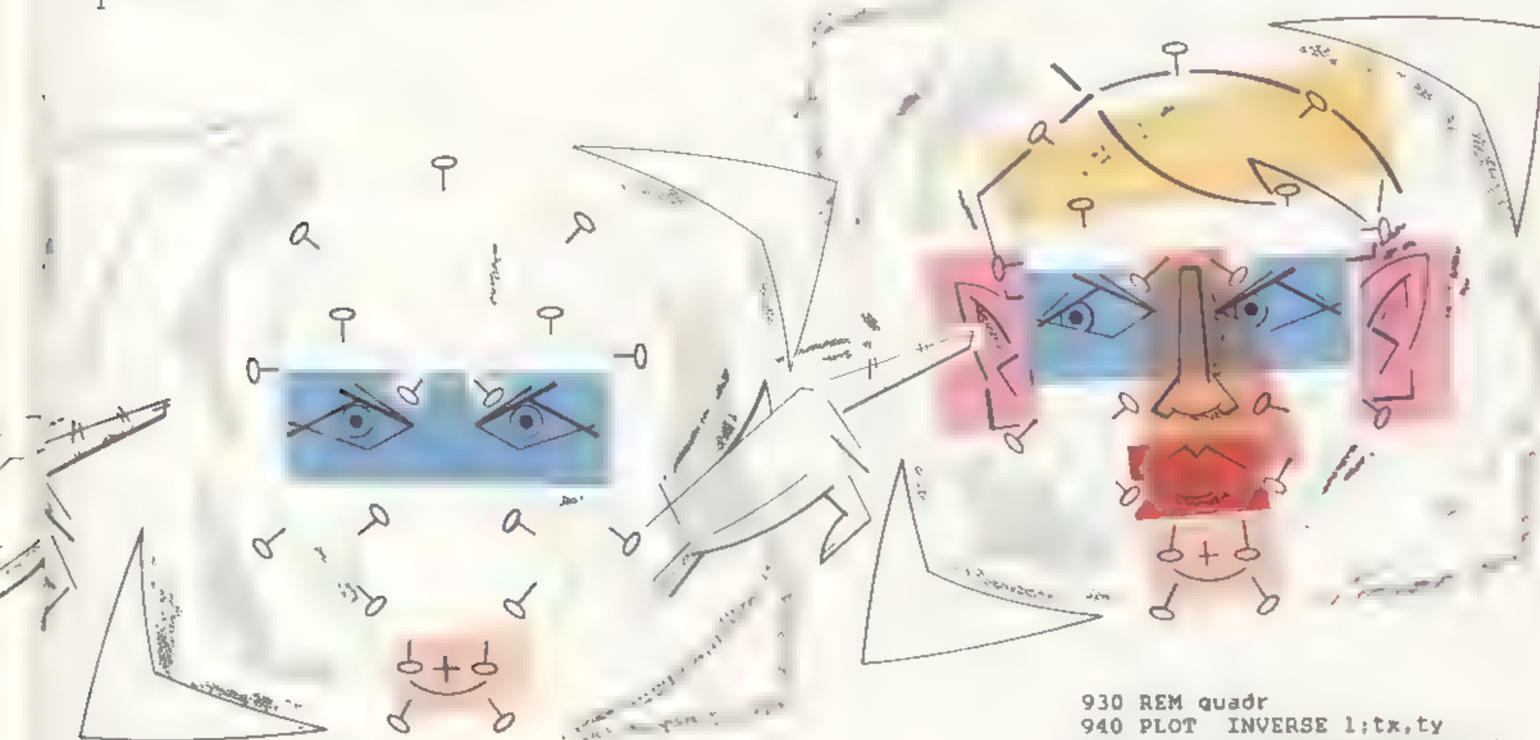


```
10 BORDER 0: PAPER 0: INK 7: OVER 0: CLS
15 LET inicio=100: LET pick=270: LET posicao=430: LET modifica=490: LET seta=390: LET desenha=640: LET fixa=700: LET limpa=750: LET checa=820: LET triang=860: LET quadr=930: LET retang=1010: LET pent=1090: LET hex=1180
17 GOSUB limpa
```



```
 40 GOSUB pick
 50 GOSUB posicao
 60 GOSUB modifica
 70 IF INKEY$<>"q" THEN  GOTO 40
 80 STOP
110 OVER 0
120 LET x=30: LET y=50: LET phi=0: LET scal=1
130 LET mex=120: LET flag=0: LET selec=0
140 LET c=scal*COS (phi)
150 LET s=scal*SIN (phi)
170 LET tx=32: LET ty=25: GOSUB triang
190 LET tx=70: LET ty=25: GOSUB quadr
210 LET tx=120: LET ty=25: GOSUB retang
220 LET tx=180: LET ty=25: GOSUB pent
230 LET tx=230: LET ty=25: GOSUB hex
240 OVER 1
```

```
250 PRINT  OVER 1; INK 6;AT 20
,mex/8;"-"
260 RETURN
300 GOSUB seta
310 GOSUB checa
320 IF CODE INKEY$=8 THEN  LET
mex=mex-8
330 IF CODE INKEY$=9 THEN  LET
mex=mex+8
340 GOSUB seta
350 IF INKEY$="s" THEN  LET fl
ag=1
360 IF flag=0 THEN  GOTO 300
370 GOSUB seta
380 RETURN
400 PRINT  INK 6; OVER 1;AT 20
,mex/8;"^"
410 RETURN
440 IF mex<40 THEN  LET selec=
1
```



```
450 IF mex>=40 AND mex<70 THEN
 LET selec=2
460 IF mex>=70 AND mex<150
THEN  LET selec=3
465 IF mex>=150 AND mex<180
THEN  LET selec=4
470 IF mex>=180 THEN  LET sele
c=5
480 RETURN
510 IF INKEY$="c" THEN  GOSUB
limpa: GOTO 517
515 GOSUB desenha
517 IF INKEY$=" " THEN  GOSUB
fixa: GOTO 520
518 GOSUB desenha
520 LET c=scal*COS (phi)
540 LET s=scal*SIN (phi)
550 IF INKEY$="z" THEN  LET x=
x-3
560 IF INKEY$="x" THEN  LET x=
x+3
570 IF INKEY$="k" THEN  LET y=
y+3
580 IF INKEY$="m" THEN  LET y=
y-3
590 IF INKEY$="n" THEN  LET sc
al=scal*1.1
600 IF INKEY$="j" THEN  LET sc
al=scal/1.1
610 IF INKEY$="h" THEN  LET ph
i=phi+(6*PI/180)
620 IF INKEY$="b" THEN  LET ph
i=phi (6*PI/180)
630 RETURN
645 LET tx=x: LET ty=y
650 IF selec=1 THEN  GOSUB tri
ang
660 IF selec=2 THEN  GOSUB qua
dr
670 IF selec=3 THEN  GOSUB ret
ang
675 IF selec=4 THEN  GOSUB pen
t
680 IF selec=5 THEN  GOSUB hex
685 FOR n=1 TO 30: NEXT n
690 RETURN
700 REM fixa
710 FOR n=1 TO 100: NEXT n
714 IF INKEY$="d" THEN  OVER 1
715 IF INKEY$<>"d" THEN  OVER
0
717 GOSUB desenha
720 PRINT AT 20,1;"
                         ": OVER 1
730 GOSUB inicio
740 RETURN
760 CLS
770 DRAW 0,175: DRAW 255,0:
DRAW 0,-175: DRAW -255,0
780 GOSUB inicio
790 RETURN
820 REM checa
830 IF INKEY$="c" THEN  GOSUB
limpa
840 IF INKEY$="e" THEN  STOP
850 RETURN
860 REM triang
870 PLOT  INVERSE 1;tx,ty
880 PLOT -7*s+PEEK 23677,6*c+
PEEK 23678
890 DRAW -6*c+11*s,-5*s-9*c
900 DRAW 12*c,10*s
910 DRAW -6*c-11*s,-5*s+9*c
920 RETURN
930 REM quadr
940 PLOT  INVERSE 1;tx,ty
950 PLOT 6*c-6*s+PEEK 23677,5*
s+5*c+PEEK 23678
960 DRAW  12*c,-10*s
970 DRAW 12*s,-10*c
980 DRAW 12*c,10*s
990 DRAW -12*s,10*c
1000 RETURN
1010 REM retang
1020 PLOT  INVERSE 1;tx,ty
1030 PLOT 12*c-6*s+PEEK 23677,1
0*s+5*c+PEEK 23678
1040 DRAW -24*c,-20*s
1050 DRAW 12*s,-10*c
1060 DRAW 24*c,20*s
1070 DRAW -12*s,10*c
1080 RETURN
1090 REM pent
1100 PLOT  INVERSE 1;tx,ty
1110 PLOT -10*s+PEEK 23677,9*c+
PEEK 23678
1120 DRAW -10*c+7*s,-8*s-6*c
1130 DRAW 4*c+13*s,3*s-11*c
1140 DRAW 12*c,10*s
```

```
1150 DRAW 4*c-13*s,3*s+11*c
1160 DRAW -10*c-7*s,-8*s+6*c
1170 RETURN
1180 REM hex
1190 PLOT  INVERSE 1;tx,ty
1200 PLOT -12*s+PEEK 23677,10*c
+PEEK 23678
1210 DRAW -10*c+6*s,-8*s-5*c
1220 DRAW 12*s,-10*c
1230 DRAW 10*c+6*s,8*s-5*c
1240 DRAW 10*c-6*s,8*s+5*c
1250 DRAW -12*s,10*c
1260 DRAW -10*c-6*s,-8*s+5*c
1270 RETURN
```

O programa funciona da maneira abaixo descrita:

A sub-rotina **início**, compreendida entre as linhas 110 e 260, define os valores de algumas variáveis e desenha as figuras-padrão na parte mais baixa da tela, chamando, para tanto, as sub-rotinas necessárias.

Da linha 300 à linha 380, o computador lê as teclas que movem a seta para a direita e para a esquerda e a tecla **S**, que faz com que a figura escolhida apareça no centro da tela.

As linhas 400 e 410 (sub-rotina **seta**) reimprimem a seta toda vez que uma tecla é pressionada.

As linhas 440 a 480 compõem a sub-rotina **posição**. Ela checa a posição da seta em relação às figuras e define a variável "figura".

A sub-rotina **modifica** encontra-se entre as linhas 510 e 630. A figura escolhida é redesenhada de acordo com as teclas pressionadas. Ela pode ser movimentada usando-se as teclas **Z**, **X**, **K** e **M**; **N** e **J** ampliam e reduzem a figura; **H** e **B** promovem a rotação no sentido horário e anti-horário. A barra de espaço fixa a figura na tela.

As figuras são desenhadas pela combinação das sub-rotinas **desenha**, entre as linhas 645 e 690, com as sub-rotinas que contêm a fórmula de cada uma delas. A variável "figura" indica ao computador qual destas será usada.

A sub-rotina **fixa**, entre as linhas 710 e 740, fixa a figura na tela quando a tecla **D** é apertada. Nas linhas 760 a 790 está a sub-rotina **limpa**, que é chamada quando se pressiona **C**.

A sub-rotina **checagem** — linhas 820 a 840 — verifica se a tecla **E** foi acionada para interromper o programa.

T

```
10 PCLEAR 8:PMODE 4,5:PCLS:PMOD
E 4,1:PCLS:SCREEN 1,1
20 GOSUB 1000
30 GOSUB 2000
40 GOSUB 3000
50 GOSUB 4000
70 IF INKEY$<>"Q" THEN 40
80 CLS:END
1000 X=20:Y=131:PH=0:SC=1
1010 ME=122:ST=0:V1=247:V2=253
1020 C=SC:S=0 COLOR 5
1030 X=40:Y=177:GOSUB 5000
1040 X=80:Y=174.GOSUB 5100
1050 X=120:GOSUB 5200
1060 X=160.Y=170:GOSUB 5300
1070 X=200:GOSUB 5400
1080 FOR K=1 TO 4:PCOPY K TO K+
4 NEXT
1090 X=127:Y=85
1100 RETURN
2000 COLOR 0:GOSUB 2500
2010 A$=INKEY$:GOSUB 4500
2020 IF A$=CHR$(8) AND ME>32 TH
EN ME=ME-10
```



```
2030 IF A$=CHR$(9) AND ME<212 T
HEN ME=ME+10
2040 COLOR 5:GOSUB 2500
2050 IF A$<>"S" THEN 2000
2060 COLOR 5:GOSUB 2500
2070 RETURN
2500 DRAW"BM"+STR$(ME)+",190U5N
G2F2"
2510 RETURN
3000 SL=5:IF ME<192 THEN SL=4
3010 IF ME<152 THEN SL=3
3020 IF ME<102 THEN SL=2
3030 IF ME<62 THEN SL=1
3040 RETURN
4000 ON SL GOSUB 5000,5100,5200
,5300,5400
4010 IF PEEK(339)=191 THEN PCLS
:GOSUB 1000:RETURN
4020 IF PEEK(345)=V1 THEN FOR K
=1 TO 4:PCOPY K TO K+4:NEXT:RET
URN
4030 C=SC*COS(PH)
4040 S=SC*SIN(PH)
4050 IF PEEK(343)=V1 THEN X=X-1
4060 IF PEEK(344)=V1 THEN X=X+1
4070 IF PEEK(341)=V1 THEN Y=Y-1
4080 IF PEEK(342)=V1 THEN Y=Y+1
4090 IF PEEK(343)=V2 THEN SC=SC
*1.1
4100 IF PEEK(344)=V2 THEN SC=SC
/1.1
4110 IF PEEK(341)=V2 THEN PH=PH
+ATN(1)/7.5
4120 IF PEEK(342)=V2 THEN PH=PH
-ATN(1)/7.5
4130 FOR K=1 TO 4:PCOPYK+4 TO K
:NEXT
4150 IF PEEK(338)=V1 THEN RETUR
N ELSE 4000
4500 IF A$=CHR$(12) THEN PCLS:G
OSUB 1000
4510 IF A$="Q" THEN CLS:END
4520 RETURN
5000 LINE(X-7.2*S,Y-7.2*C)-(X-6
*C+3.6*S,Y+6*S+3.6*C),PSET
5010 LINE -(X+6*C+3.6*S,Y-6*S+3
.6*C),PSET
5020 LINE-(X-7.2*S,Y-7.2*C),PSE
T
5030 RETURN
5100 LINE(X+6*C-6*S,Y-6*S-6*C)-
(X-6*S-6*C,Y+6*S-6*C),PSET
5110 LINE -(X+6*S-6*C,Y+6*S+6*C
),PSET
5120 LINE -(X+6*S+6*C,Y+6*C-6*S
),PSET
5130 LINE -(X+6*C-6*S,Y-6*S-6*C
),PSET
5140 RETURN
5200 LINE (X+12*C-6*S,Y-6*C-12*
S)-(X-12*C-6*S,Y-6*C+12*S),PSET
5210 LINE -(X-12*C+6*S,Y+6*C+12
```

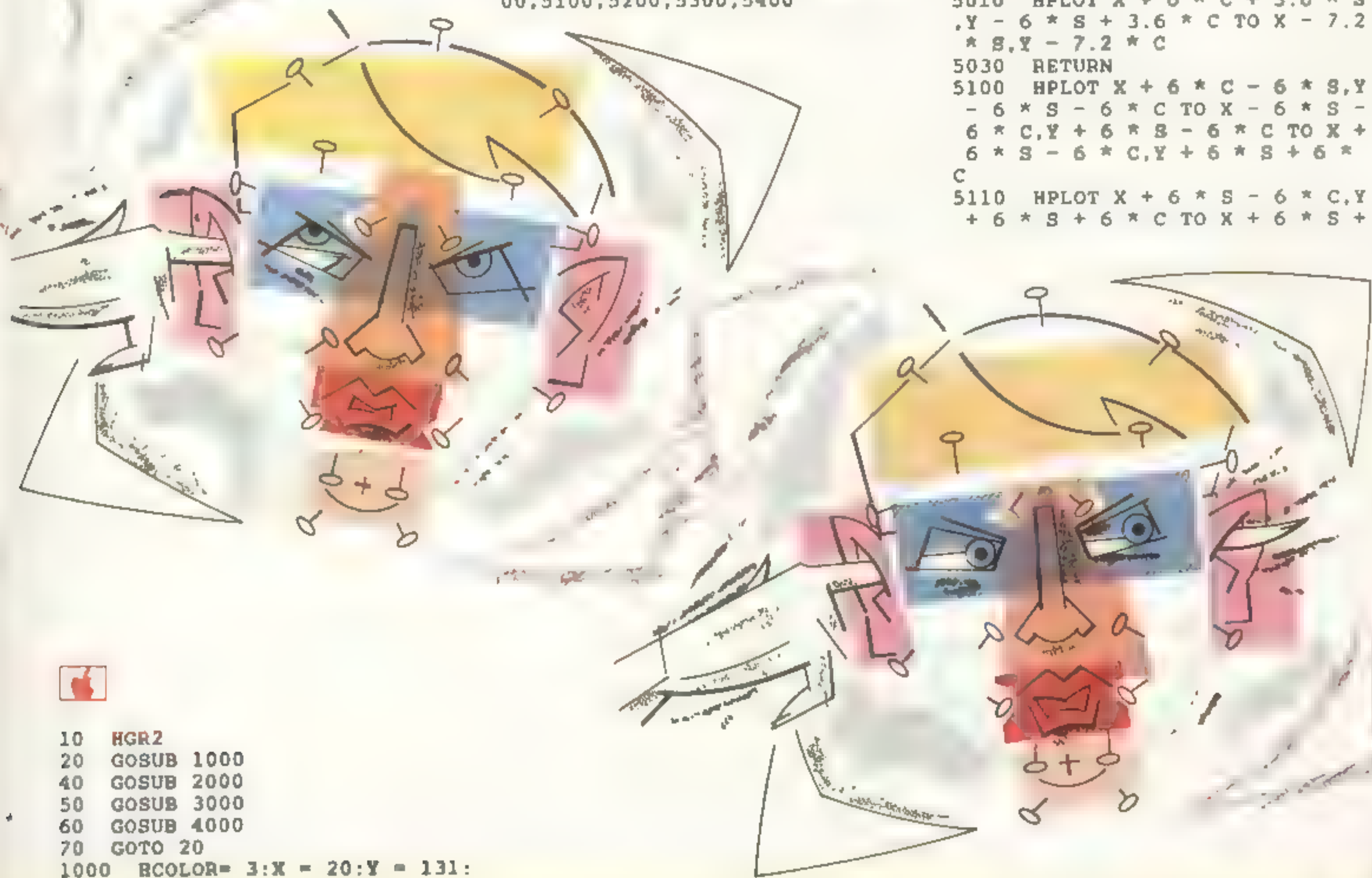
```
*S),PSET
5220 LINE -(X+12*C+6*S,Y+6*C-12
*S),PSET
5230 LINE -(X+12*C-6*S,Y-6*C-12
*S),PSET
5240 RETURN
5300 LINE (X-10.2*S,Y-10.2*C)-(
X-9.8*C-3.2*S,Y-3.2*C+9.8*S),PS
ET
5310 LINE -(X-6*C+10.2*S,Y+10.2
*C+6*S),PSET
5320 LINE -(X+6*C+10.2*S,Y+10.2
*C-6*S),PSET
5330 LINE -(X+9.8*C-3.2*S,Y-3.2
*C-9.8*S),PSET
5340 LINE -(X-10.2*S,Y-10.2*C),
PSET
5350 RETURN
5400 LINE(X-12*S,Y-12*C)-(X-10.
4*C-6*S,Y-6*C+10.4*S),PSET
5410 LINE -(X-10.4*C+6*S,Y+6*C+
10.4*S),PSET
5420 LINE -(X+12*S,Y+12*C),PSET
5430 LINE -(X+10.4*C+6*S,Y+6*C-
10.4*S),PSET
5440 LINE -(X+10.4*C-6*S,Y-6*C-
10.4*S),PSET
5450 LINE -(X-12*S,Y-12*C),PSET
5460 RETURN
```

```
10  HGR2
20  GOSUB 1000
40  GOSUB 2000
50  GOSUB 3000
60  GOSUB 4000
70  GOTO 20
1000  HCOLOR= 3:X = 20:Y = 131:
PH = 0:SC = 1
1010 ME = 122
1020 C = SC:S = 0
1030 X = 40:Y = 177: GOSUB 5000

1040 X = 80:Y = 174: GOSUB 5100

1050 X = 120: GOSUB 5200
1060 X = 160:Y = 170: GOSUB 530
0
1070 X = 200: GOSUB 5400
1090 X = 127:Y = 85
1100  RETURN
2000  HCOLOR= 3: GOSUB 2500
2005  GET A$
2010  HCOLOR= 0: GOSUB 2500
2020  IF A$ = "Z" AND ME > 32 T
HEN ME = ME - 10
2030  IF A$ = "X" AND ME < 212
THEN ME = ME + 10
2050  IF A$ <  > "S" THEN 2000
2070  RETURN
2500  HPLOT ME,190 TO ME,182 TO
 ME - 5,185: HPLOT ME,182 TO ME
 + 5,185
2510  RETURN
3000 SL = 5: IF ME < 192 THEN S
L = 4
3010  IF ME < 152 THEN SL = 3
3020  IF ME < 102 THEN SL = 2
3030  IF ME < 62 THEN SL = 1
3040  RETURN
4000  HCOLOR= 3: ON SL GOSUB 50
00,5100,5200,5300,5400
4010  GET A$
4015  HCOLOR= 0: ON SL GOSUB 50
00,5100,5200,5300,5400
4020  IF A$ = " " THEN  HCOLOR=
 3: ON SL GOSUB 5000,5100,5200,
5300,5400: RETURN
4030 C = SC *  COS (PH)
4040 S = SC *  SIN (PH)
4050  IF A$ = "Z" THEN X = X -
3
4060  IF A$ = "X" THEN X = X +
3
4070  IF A$ = ";" THEN Y = Y -
3
4080  IF A$ = "." THEN Y = Y +
3
4090  IF A$ = "M" THEN SC = SC
* 1.1
4100  IF A$ = "N" THEN SC = SC
/ 1.1
4110  IF A$ = "K" THEN PH = PH
+  ATN (1) / 7.5
4120  IF A$ = "L" THEN PH = PH
-  ATN (1) / 7.5
4140  IF A$ = "C" THEN  GOTO 10

4145  IF A$ = "F" THEN  TEXT :
END
4150  GOTO 4000
5000  HPLOT X - 7.2 * S,Y - 7.2
 * C TO X - 6 * C + 3.6 * S,Y +
 6 * S + 3.6 * C TO X + 6 * C +
 3.6 * S,Y - 6 * S + 3.6 * C
5010  HPLOT X + 6 * C + 3.6 * S
,Y - 6 * S + 3.6 * C TO X - 7.2
 * S,Y - 7.2 * C
5030  RETURN
5100  HPLOT X + 6 * C - 6 * S,Y
 - 6 * S - 6 * C TO X - 6 * S -
 6 * C,Y + 6 * S - 6 * C TO X +
 6 * S - 6 * C,Y + 6 * S + 6 *
C
5110  HPLOT X + 6 * S - 6 * C,Y
 + 6 * S + 6 * C TO X + 6 * S +
```



```
 6 * C,Y + 6 * C - 6 * S TO X +
 6 * C - 6 * S,Y - 6 * S - 6 *
C
5140  RETURN
5200  HPLOT X + 12 * C - 6 * S,
Y - 6 * C - 12 * S TO X - 12 *
C - 6 * S,Y - 6 * C + 12 * S TO
 X - 12 * C + 6 * S,Y + 6 * C +
 12 * S
5210  HPLOT X - 12 * C + 6 * S,
Y + 6 * C + 12 * S TO X + 12 *
C + 6 * S,Y + 6 * C - 12 * S TO
 X + 12 * C - 6 * S,Y - 6 * C -
 12 * S
5240  RETURN
5300  HPLOT X - 10.2 * S,Y - 10
.2 * C TO X - 9.8 * C - 3.2 * S
,Y - 3.2 * C + 9.8 * S TO X - 6
 * C + 10.2 * S,Y + 10.2 * C +
6 * S
5310  HPLOT X - 6 * C + 10.2 *
S,Y + 10.2 * C + 6 * S TO X + 6
 * C + 10.2 * S,Y + 10.2 * C -
6 * S TO X + 9.8 * C - 3.2 * S,
Y - 3.2 * C - 9.8 * S
5320  HPLOT X + 9.8 * C - 3.2 *
 S,Y - 3.2 * C - 9.8 * S TO X -
 10.2 * S,Y - 10.2 * C
5350  RETURN
5400  HPLOT X - 12 * S,Y - 12 *
 C TO X - 10.4 * C - 6 * S,Y -
6 * C + 10.4 * S TO X - 10.4 *
C + 6 * S,Y + 6 * C + 10.4 * S
TO X + 12 * S,Y + 12 * C
5410  HPLOT X + 12 * S,Y + 12 *
 C TO X + 10.4 * C + 6 * S,Y +
6 * C - 10.4 * S TO X + 10.4 *
C - 6 * S,Y - 6 * C - 10.4 * S
5420  HPLOT X + 10.4 * C - 6 *
S,Y - 6 * C - 10.4 * S TO X - 1
2 * S,Y - 12 * C
5460  RETURN
```



O programa do TK-2000 é igual ao do Apple, com estas modificações:

```
2020  IF A$ =  CHR$ (8) AND ME
> 32 THEN ME = ME - 10
2030  IF A$ =  CHR$ (21) AND ME
 < 212 THEN ME = ME + 10
4050  IF A$ =  CHR$ (8) THEN X
= X - 3
4060  IF A$ =  CHR$ (21) THEN X
 = X + 3
4070  IF A$ =  CHR$ (112) THEN
Y = Y - 3
4080  IF A$ =  CHR$ (113) THEN
Y = Y + 3
```



```
10 SCREEN 2
20 GOSUB 1000
40 GOSUB 2000
50 GOSUB 3000
60 GOSUB 4000
70 GOTO 20
1000 COLOR 14:X=20:Y=131:PH=0:S
C=1
1010 ME=122
1020 C=SC:S=0
1030 X=40:Y=177:GOSUB 5000
1040 X=80:Y=174:GOSUB 5100
1050 X=120:GOSUB 5200
1060 X=160:Y=170:GOSUB 5300
1070 X=200:GOSUB 5400
1090 X=127:Y=85
1100 RETURN
2000 COLOR 14:GOSUB 2500
2005 A$=INKEY$:IF A$="" THEN 20
05
2010 COLOR 4:GOSUB 2500
2020 IF A$=CHR$(29) AND ME>32 T
HEN ME=ME-10
2030 IF A$=CHR$(28) AND ME<212
THEN ME=ME+10
2050 IF A$<>"S" THEN 2000
2070 RETURN
2500 DRAW"BM"+STR$(ME)+",190U5N
G2F2"
2510 RETURN
3000 SL=5:IF ME<192 THEN SL=4
3010 IF ME<152 THEN SL=3
3020 IF ME<102 THEN SL=2
3030 IF ME<62 THEN SL=1
3040 RETURN
4000 COLOR 14:ON SL GOSUB 5000,
5100,5200,5300,5400
4010 A$=INKEY$:IF A$="" THEN 40
10
4015 COLOR 4:ON SL GOSUB 5000,5
100,5200,5300,5400
4020 IF A$=" " THEN COLOR 14:ON
 SL GOSUB 5000,5100,5200,5300,5
400:RETURN
4030 C=SC*COS(PH)
4040 S=SC*SIN(PH)
4050 IF A$=CHR$(29) THEN X=X-3
4060 IF A$=CHR$(28) THEN X=X+3
4070 IF A$=CHR$(30) THEN Y=Y-3
4080 IF A$=CHR$(31) THEN Y=Y+3
4090 IF A$="M" THEN SC=SC*1.1
4100 IF A$="N" THEN SC=SC/1.1
4110 IF A$="K" THEN PH=PH+ATN(1
)/7.5
4120 IF A$="L" THEN PH=PH-ATN(1
)/7.5
4140 IF A$="C" THEN CLS:GOTO 20
4145 IF A$="F" THEN COLOR 15:EN
D
4150 GOTO 4000
5000 LINE (X-7.2*S,Y-7.2*C)-(X-
6*C+3.6*S,Y+6*S+3.6*C)
5010 LINE -(X+6*C+3.6*S,Y-6*S+3
.6*C)
5020 LINE -(X-7.2*S,Y-7.2*C)
5030 RETURN
5100 LINE (X+6*C-6*S,Y-6*S-6*C)
-(X-6*S-6*C,Y+6*S-6*C)
5110 LINE -(X+6*S-6*C,Y+6*S+6*C
)
5120 LINE -(X+6*S+6*C,Y+6*C-6*S
)
5130 LINE -(X+6*C-6*S,Y-6*S-6*C
)
5140 RETURN
5200 LINE (X+12*C-6*S,Y-6*C-12*
S)-(X-12*C-6*S,Y-6*C+12*S)
5210 LINE -(X-12*C+6*S,Y+6*C+12
*S)
5220 LINE -(X+12*C+6*S,Y+6*C-12
*S)
5230 LINE -(X+12*C-6*S,Y-6*C-12
*S)
5240 RETURN
5300 LINE (X-10.2*S,Y-10.2*C)-(
X-9.8*C-3.2*S,Y-3.2*C+9.8*S)
5310 LINE -(X-6*C+10.2*S,Y+10.2
*C+6*S)
5320 LINE -(X+6*C+10.2*S,Y+10.2
*C-6*S)
5330 LINE -(X+9.8*C-3.2*S,Y-3.2
*C-9.8*S)
5340 LINE -(X-10.2*S,Y-10.2*C)
5350 RETURN
5400 LINE (X-12*S,Y-12*C)-(X-10
.4*C-6*S,Y-6*C+10.4*S)
5410 LINE -(X-10.4*C+6*S,Y+6*C+
10.4*S)
5420 LINE -(X+12*S,Y+12*C)
5430 LINE -(X+10.4*C+6*S,Y+6*C-
10.4*S)
5440 LINE -(X+10.4*C-6*S,Y-6*C-
10.4*S)
5450 LINE -(X-12*S,Y-12*C)
5460 RETURN
```

A inicialização encontra-se entre as linhas 1000 e 1100, onde uma série de variáveis são definidas. As linhas 2000 a 2070 compreendem a sub-rotina encarregada de selecionar a figura. A seta é movimentada para a direita e para a esquerda através das teclas de controle (o Apple usa as mesmas teclas do programa anterior).

As linhas 3000 a 3040 identificam a figura que está sendo apontada pela seta, de acordo com a variável **ME**. Ao pressionar a tecla **S**, a figura escolhida aparecerá no centro da tela.

A sub-rotina de desenho situa-se entre as linhas 4000 e 4150. A linha 4000 desvia o programa para a sub-rotina que contém as instruções para desenhar a figura escolhida. A linha 4020 fixa a figura na tela se a barra de espaço for acionada.

As linhas 4090 e 4100 verificam se as teclas **M** e **N** foram pressionadas, ampliando ou reduzindo o desenho. A opção final é a rotação — linhas 4110 e 4120. As teclas **K** e **L** controlam a direção (para a esquerda e para a direita) em que será executada.

Ao pressionar **C** (**<CLEAR>** no TRS-Color), a tela será apagada, ficando pronta para o desenho seguinte.

As sub-rotinas restantes contêm instruções para a construção das cinco figuras. As linhas 5000 a 5030 desenham o triângulo; as linhas 5100 a 5140 traçam o quadrado; as linhas 5200 a 5240, o retângulo; as linhas 5300 a 5350, o pentágono e as linhas 5400 a 5460, o hexágono.

Tome cuidado para que seu desenho não saia da tela, o que poderia acarretar a interrupção do programa e uma mensagem de erro. Para interromper o programa, pressione **F**.

EFEITOS SONOROS COMPLEXOS

- COMO OBTER O EFEITO DE UM "TIRO LASER"
- MOVIMENTOS DO CONE
- PAUSAS
- CONTROLE DA DURAÇÃO

Os usuários do Apple e do TK-2000 já aprenderam a utilizar rotinas em código de máquina para produzir sons em seu micro. Verão agora como criar efeitos sonoros mais complexos.

No artigo da página 712, explicamos como contornar as limitações dos micros das linhas Apple e TK-2000 para que produzam sons. Examinaremos aqui a criação de efeitos mais sofisticados.

Com a rotina que se segue obtemos um ruído que pode ser utilizado como "tiro laser" em um jogo de ação. O truque consiste em combinar um som que vai se tornando mais agudo com outro que fica cada vez mais grave.



```
10 ORG 800
20 LDA #$00
30 STA $FF
40 LDA #$00
50 STA $FE
60 LOOP LDA #$00
70 STA $C030
80 LDX $FF
90 PAUSEA NOP
100 NOP
110 NOP
120 NOP
130 DEX
140 BNE PAUSEA
150 INC $FF
160 LDA #$00
170 STA $C030
180 LDX $FF
190 PAUSEB NOP
200 NOP
210 NOP
220 NOP
230 DEX
240 BNE PAUSEB
250 DEC $FE
260 BEQ FIM
270 JMP LOOP
280 FIM RTS
290 END
```



Para o míni-Assembler do TK-2000, a listagem é a seguinte:

```
0320-    LDA    #$00
0322-    STA    $FF
0324-    LDA    #$00
0326-    STA    $FE
0328-    LDA    #$00
032A-    STA    $C030
032D-    LDX    $FF
032F-    NOP
0330-    NOP
0331-    NOP
0332-    NOP
0333-    DEX
0334-    BNE    $032F
0336-    INC    $FF
0338-    LDA    #$00
033A-    STA    $C030
033D-    LDX    $FF
033F-    NOP
0340-    NOP
0341-    NOP
0342-    NOP
0343-    DEX
0344-    BNE    $033F
0346-    DEC    $FE
0348-    BEQ    $034D
034A-    JMP    $0328
034D-    RTS
```

Depois de estabelecido o endereço inicial, as quatro primeiras instruções colocam o valor zero nas posições $FF e $FE, que servirão como contadores. O registro A é usado para isso, porque não há no chip 6502 uma instrução que coloque um certo valor diretamente em um endereço de memória.

EMISSÃO DE SOM

A seguir, o registro A é novamente carregado com zero e a instrução **STA $C030** produz um movimento no cone do alto-falante.

O valor contido na posição de memória $FF é, então, colocado no registro X e as seis linhas seguintes provocam uma pausa com duração proporcional àquele valor. Como no programa anterior, a linha 130 encarrega-se de subtrair uma unidade de X e, enquanto este contador não tiver sido reduzido a zero, o laço entre as linhas 90 e 140 vai sendo repetido.

Após essa pausa, a instrução **INC $FF** soma a unidade ao conteúdo da posição $FF, a fim de que a pausa seguinte seja maior.

As linhas 160 e 170 produzem novamente um movimento do alto-falante e, logo depois, as linhas 190 a 240 provocam nova pausa, agora com duração proporcional ao conteudo de $FE.

Ao contrário do que aconteceu na linha 150, a instrução **DEC $FE** subtrai uma unidade do conteúdo da posição $FE, fazendo com que, na próxima vez, a pausa seja menor.

FIM DA ROTINA

A instrução **BEQ FIM** termina o programa quando o conteúdo de $FE for reduzido a zero. O final da rotina ocorre com o desvio para o rótulo FIM, que contém a instrução **RTS**. Enquanto houver um número maior que zero em $FE, a instrução **JMP LOOP** faz o laço entre as linhas 60 e 270 ser repetido.

PAUSEA E PAUSEB

Como podemos observar, o programa produz movimentos do alto-falante — e, portanto, emite som — em duas ocasiões: nas linhas 70 e 170. Entre esses dois eventos ocorrem duas pausas: uma que é rotulada **PAUSEA** e a outra, **PAUSEB**. **PAUSEA** é controlada por $FF e vai aumentando de tamanho; **PAUSEB**, por $FE e vai diminuindo de tamanho. Combinam-se, dessa maneira, dois sons, um com freqüência crescente e outro com freqüência decrescente.

Na realidade, o laço **PAUSEA** completa 256 voltas na primeira vez, dando uma volta a mais cada vez que e executado. Como o contador $FF só usa um byte, 256 é representado pelo numero zero, e 0 + 1 = 1. Assim, **PAUSEA** dá uma volta na segunda execução e uma volta a mais a cada nova execução. Já **PAUSEB** começa com 256 voltas e vai dando uma volta a menos a cada execução.

CONTROLE DA DURAÇÃO

Para diminuir a duração do efeito sonoro, basta retirar alguns comandos **NOP**. Retire quantidades iguais de cada laço de pausa, pois, do contrário, o timbre do som será alterado. Quem estiver usando o míni-Assembler precisará recalcular os desvios.

AVALANCHE: O VÔO DAS GAIVOTAS

PREPARANDO O VÔO





DECOLAGEM



A listagem que você digitará a seguir não faz parte do programa principal. Este é composto por quatro rotinas, três das quais já publicadas. Apresentaremos a última delas no artigo que encerra a série *Avalanche*.

Este programa e todos os que o seguem, a partir do presente artigo, constituem sub-rotinas isoladas que irão ser chamadas pela quarta rotina do programa principal. Como esta será montada na seqüência da terceira rotina, a listagem que fornecemos aqui começa no endereço −11380. Deixamos, assim, um espaço reservado para a última parte do programa principal do jogo.

Esta rotina, além de colocar as gaivotas na tela, faz com que elas batam as asas para cima e para baixo usando só duas estruturas de animação.

```
10    org 54156
20    gul ld a,(-5206)
30    inc a
40    res 3,a
50    ld (-5206),a
60    ld b,28
70    cp 4
80    jr c,gpt
90    ld b,32
100    gpt push bc
110     ld a,b
120     ld hl,(62407)
130     ld de,42
140    add hl,de
150    push hl
160    push af
170    call 77
180    pop af
190    pop hl
200    inc hl
210    add a,2
220    call 77
230    pop bc
240    ld a,b
250    ld hl,(62407)
260    ld de,68
270    add hl,de
280    push hl
290    push af
300    call 77
310    pop af
320    pop hl
330    inc hl
340    add a,2
350    call 77
360    ret
370    end
```

DEFINIÇÃO DO VÔO

A posição de memória −5206 contém a variável de atraso da gaivota, que tem a função de impedir que as aves batam as asas muito rapidamente. O atraso completo é carregado e incrementado no acumulador A.

Para que as asas se movimentem para cima e para baixo de maneira adequada, o atraso da gaivota deve oscilar numa faixa restrita. A instrução **res 3,a** se encarrega disso, ajustando o bit 3 do acumulador com o valor 0. Analisando essa instrução, verifica-se que a variável de atraso será reajustada ao valor 0 sempre que ela for incrementada com um valor superior a 7. Nesse caso, o resultado do incremento e do ajuste para zero é armazenado novamente no endereço −5206.

Para dar a impressão de que a gaivota está batendo as asas, utilizamos padrões de bits para duas gaivotas — uma com as asas para cima e outra com as asas para baixo.

Os códigos dos padrões para a primeira gaivota são 28 e 30, e para a segunda, 32 e 34. Cada padrão compreende oito bytes de dados e cada gaivota possui dois padrões.

O registro B é carregado com o código do primeiro padrão da primeira gaivota. Depois, o conteúdo do acumulador — onde ainda está o atraso da gaivota — é comparado com o número 4.

Uma instrução **cp** equivale a uma subtração cujo resultado não foi armazenado. O número 4 é subtraído do conteúdo do acumulador, mas não se armazena o resultado em nenhum lugar. Essa operação tem a função exclusiva de ajustar os indicadores ou balizas (**flags**). Se o atraso da gaivota for menor do que 4, o indicador chamado **carry** é ajustado com o valor 1; se for maior ou igual a 4, **carry** não é ajustado.

A instrução **jr, c,gpt** faz o processador saltar para o rótulo **gpt** se **carry** foi

ajustado com 1. Assim, se o atraso da gaivota for menor do que 4, o processador salta a instrução seguinte e o código 28 permanece em B.

Se o atraso da gaivota for maior ou igual a 4, **carry** não foi ajustado e o salto não ocorre. O registro B é, então, carregado com o código 32, correspondente ao primeiro padrão da segunda gaivota na tabela de padrões.

DECOLAGEM

A rotina **gpt** — incumbida de colocar as duas gaivotas na tela — começa armazenando na pilha o código do primeiro padrão da gaivota que está sendo impressa. Isso é feito porque, como temos que imprimir a gaivota duas vezes, esse código sofrerá alteração durante a primeira impressão.

Para imprimir uma gaivota na tela, recorremos à rotina 77 da ROM, já utilizada em partes anteriores do jogo. Antes, precisamos ajustar seus parâmetros. O acumulador deve conter o código do padrão e o par de registros HL, o endereço na tabela de nomes onde vamos colocá-lo — ou seja, a posição de impressão na tela.

O código do primeiro padrão da gaivota é transferido do registro B para o acumulador. Em seguida, o par HL é carregado com o endereço inicial da tabela de nomes, que está armazenado nos endereços 62407 e 62408 da RAM. Adicionando-se o número 42 a esse endereço, por meio do par de registros DE, obtém-se a posição na tela da metade inicial da primeira gaivota.

Os conteúdos do par de registros HL e do acumulador são preservados na pilha, uma vez que podem ser alterados pela rotina 77. Esta é chamada logo em seguida e imprime a metade esquerda da primeira gaivota.

O apontador da posição na tela é recuperado da pilha, voltando ao par HL, enquanto o código do padrão retorna ao acumulador. Em seguida, o apontador em HL é incrementado para indicar a posição de impressão da metade direita. O número 2 é então adicionado ao acumulador, que passa a conter o código do padrão da segunda metade. Quando a primeira gaivota já está na tela, a rotina 77 é chamada.

O código do padrão da metade esquerda da gaivota é recuperado da pilha para o registro B, uma vez que essa mesma figura será novamente impressa em outra posição.

O procedimento é análogo ao já descrito, só que a adição do número 68 ao endereço inicial da tabela de nomes em HL indica a posição de impressão da segunda gaivota. A rotina 77 é chamada duas vezes, uma para cada metade, e o endereço da segunda gaivota se completa na tela também.

TESTANDO

Para testar a listagem aqui apresentada, espere a publicação da última rotina da série *Avalanche*. Se quiser se antecipar, digite uma pequena rotina em código de máquina que chame a rotina deste artigo várias vezes, com um pequeno atraso entre cada chamada. Tal atraso pode ser provocado por um laço qualquer. Depois disso, carregue a tabela de padrões e veja as gaivotas batendo suas asas.



SONS E RUÍDOS NO TRS-80

O TRS-80 não conta com comandos para a produção de efeitos sonoros. Dois pequenos programas em código de máquina, porém, permitem a criação de diversos sons nesse micro.

Muitos tipos de jogos e outros programas se tornarão bem mais interessantes e "profissionais" se você adicionar a eles música e efeitos sonoros, como ruídos de tiros, explosões, disparos de canhões de laser e coisas do gênero. Na verdade, esta é uma das razões pelas quais a maioria dos microcomputadores mais modernos tem comandos especiais em BASIC para o controle de sintetizadores internos de som, como é o caso do TRS-Color, do Spectrum, do MSX e do TK-2000.

Os micros compatíveis com a linha TRS-80 não contam com esse recurso, tão simples de usar. Mas isso não quer dizer que seja impossível programar efeitos sonoros e musicais nessas máquinas. Conhecendo alguns truques em linguagem Assembler, é fácil escrever pequenas rotinas para adicionar som aos seus programas.

FAÇA UM SOM

Fazer o programa não é tudo, porém. Normalmente, os microcomputadores da linha TRS-80 não têm alto-falante interno, como ocorre com o Apple, e nem possuem linha de conexão para o alto-falante da TV ou do monitor, como é o caso do MSX e do TK-2000.

Por essa razão, se quisermos produzir sons nesse micro, devemos utilizar a saída para o gravador cassete. Esse recurso vale também para os modelos da linha TRS-80 que dispõem de alto-falante interno (por exemplo, o Prológica CP-500), ligado em paralelo com a saída para cassete.

PORTA DE SAÍDA

Como é possível produzir efeitos sonoros numa saída desse tipo?

Um gravador de áudio é capaz de registrar apenas sons na faixa audível (entre 200 e 10.000 Hz, com os valores máximo e mínimo dependendo da qualidade do aparelho). Ora, uma interface especial na saída para o gravador transforma os impulsos binários (0 e 1), que constituem a "linguagem" interna do computador, em ondas elétricas com freqüência compatível. Em outras palavras, se você colocar uma fita gravada com um programa de computador para escutar, ouvirá uma "sinfonia" musical caótica, que corresponde aos sinais enviados pelo computador.

Essa interface de conversão digital-analógica está conectada internamente a uma *porta* do computador.

Uma porta é um canal de comunicação entre o microprocessador e o mundo externo. Em geral, uma porta faz a comunicação nos dois sentidos (entrada e saída), mas muitos micros usam determinadas portas só para um tipo de operação. É o caso da porta para gravador cassete, que é apenas de saída.

Em consequência, tudo o que necessitamos fazer para ouvir sons produzidos pela porta do gravador cassete é conectar o plugue do fio que normalmente é utilizado para gravar programas e dados (e que é chamado de AUX ou MIC) a um sistema de som, dotado de amplificador e alto-falante.

Quem tem um microcomputador da linha TRS-80, ou compatível, já dotado de alto-falante interno, não precisa nem mesmo tomar essas medidas.

SONS POR SOFTWARE

Para acionar uma porta de saída, usamos a instrução **OUT**, em linguagem Assembler, que se encarrega de enviar um byte para essa porta.

Para gerar sons com a instrução **OUT**, recorremos à velocidade do código de máquina. Tudo o que é preciso fazer, neste caso, é movimentar alternadamente o cone do alto-falante para fora e para dentro. Se fizermos isso uma vez, produziremos um clique do tipo que costuma ocorrer quando ligamos um aparelho de som. O truque consiste em repetir a operação seguidas vezes, e de modo bastante rápido. Se os movimentos forem suficientemente velozes, a sucessão de cliques vai parecer um zumbido. Quanto mais acelerado for o movimento de vaivém do cone, mais agudo será o som obtido.

Para escrever um programa destinado a criar qualquer tipo de efeito sonoro, é necessário que se conheçam alguns fatos básicos sobre a porta do gravador do micro TRS-80:

- A porta de saída do gravador tem o endereço 255 (FF em hexadecimal).
- Apenas o primeiro e o segundo bits do byte enviado à porta de saída são usados pela interface de conversão. O bit zero determina a saída de um impulso de som, ao passo que o bit um controla o motor do gravador.
- Quando o bit zero da porta é igualado a 1, ocorre um impulso audível positivo na linha AUX ou MIC do gravador. Quando o bit zero é igualado a 0, ocorre um impulso negativo.

Portanto, para conseguir uma aproximação de uma onda sinusoidal (tom puro de uma certa freqüência) na porta de saída, o software deve fazer o seguinte: um impulso positivo é enviado, e um pequeno período de espera é introduzido. Em seguida, um impulso negativo é enviado, e novo período de espera deve ocorrer.

Ao ser aumentado o período de espera, a frequência do som gerado sofrerá uma diminuição; se, ao contrário, esse período for reduzido, a frequência obtida será incrementada.

Para produzir um ruído ou qualquer outro efeito sonoro diferente, basta variar os dois intervalos de espera segundo algum padrão (ou, então, aleatoriamente: com isso conseguiremos o chamado "ruído branco", semelhante ao chiado de um rádio).

TONS PUROS

Apresentamos a seguir uma sub-rotina em linguagem de máquina que possibilita a produção de tons puros (bipes), com freqüência e duração individualmente controláveis.

Para facilitar o uso dessa rotina por programas em BASIC, a sub-rotina é inteiramente realocável, e é colocada em uma parte protegida da memória por uma rotina de inicialização, em BASIC, que precisa ser chamada apenas uma vez. Uma segunda rotina, também em BASIC, invoca a sub-rotina em código de máquina, por meio do comando **USR**.

É possível armazenar a rotina em có-

digo de máquina em uma parte protegida da memória. Para isso, reinicialize o computador e forneça o limite mínimo da memória protegida quando ele apresentar na tela a questão "MEM USADA?" ou "MEMSIZE?". No exemplo a seguir, imaginamos que o computador tem 32Kbytes, e fixamos o limite mínimo da memória; assim, a rotina passará a ser armazenada na locação 49001. Para usar outro limite de memória, altere o valor da variável **IM** na linha 1010 do programa de inicialização. O valor sugerido para um computador com 48Kbytes de RAM seria 65000.

A versão aqui apresentada funciona em microcomputadores da linha TRS-80 com BASIC Nível II (para cassete):

```
1000 ' ROTINA DE INICIALIZACAO
1010 IM=49000
1020 FOR I=1 TO 35
1030 READ N:POKE IM+I,N
1040 NEXT N
1050 DATA 205,127,10,14,255,6
1060 DATA 1,237,65,17,29,0,27
1070 DATA 122,179,32,251,6,2
1080 DATA 237,65,17,29,0,27
1090 DATA 122,179,32,251,43
1100 DATA 124,181,32,227,201
1110 POKE 16527,INT(IM/256) :
POKE 16526,IM-INT(IM/256)*256+1
1120 RETURN
```

A rotina a ser chamada no momento de produção do som é assim:

```
2000 ' SUBROTINA PARA SOM
2010 CY=FQ*DR:N=29483/FQ
2020 POKE IM+11,N:POKE IM+23,N
2030 I=USR(CY):RETURN
```

Note que a rotina exige dois argumentos, que devem ser fornecidos pelo programa que a chamou: **DR** é a duração, em segundos, e **FQ**, a freqüência do som, em hertz (ciclos por segundo). Apresentamos a seguir um pequeno programa de teste, que exemplifica a maneira de usar as rotinas anteriores.

```
10 'PROGRAMA DE TESTE
20 GOSUB 1000 :'INICIALIZACAO
30 CLS:INPUT"DURACAO (S)";DR
40 INPUT"FREQUENCIA (HZ)";FQ
50 GOSUB 2000:GOTO 30
```

Para rodar o programa em micros com Disk BASIC (para disquete), faça as seguintes modificações:

```
1110 DEFUSR0=IM+1
2040 I=USR0(CY):RETURN
```

Os programas apresentados funcionam da seguinte maneira: a rotina de inicialização lê os códigos decimais correspondentes à rotina em linguagem de máquina, e os coloca na memória protegida por meio do comando **POKE**.

Antes de terminar, a rotina define o endereço de partida da rotina de máquina, que é, então, comunicado ao interpretador (par de apontadores 16526 e 16527, na versão destinada a gravador cassete, e DEFUSR, na versão adaptada para disco).

Na pequena rotina de produção de sons, inicialmente é calculado o número total de ciclos (positivo/negativo) necessário para produzir a duração pedida. Esse valor é armazenado na variável **CY**, que é passada à rotina **USR** por meio de seu argumento.

A duração total da pausa a ser executada entre os ciclos (**N**) é também calculada e fornecida à rotina diretamente, mediante dois comandos **POKE**, nos pontos onde é necessária.

A última linha da rotina em linguagem BASIC se encarrega de chamar a rotina em código de máquina, e retorna depois de ser executada.

TIROS, EXPLOSÕES E CHIADOS

O próximo programa funciona segundo o mesmo princípio que o anterior, só que o intervalo entre as fases positiva e negativa da onda sonora é alterado aleatoriamente a cada instante. Ao invés de um som puro (freqüência constante em toda a duração), esse tipo de intervalo produz um impulso de "ruído branco", que tem como característica uma mistura de todas as freqüências (daí seu nome, em analogia com a cor branca). Portanto, resta ao software controlar apenas a duração total do impulso sonoro. Durações muito curtas produzem um efeito sonoro semelhante a um tiro; durações um pouco maiores, algo semelhante a uma explosão, e durações muito longas, um chiado que lembra a estática de rádio (ou, se você quiser, o barulho produzido pelo reator de um avião a jato). Todos esses efeitos têm suas aplicações em programas de jogos.

FUNCIONAMENTO DA ROTINA

Da mesma maneira que a anterior, a sub-rotina é inteiramente realocável, e é colocada em uma parte protegida da memória por uma rotina de inicialização em BASIC, que precisa ser chamada apenas uma vez. Uma segunda rotina, também em BASIC, invoca a sub-rotina em código de máquina, por intermédio do comando **USR**.

A versão a seguir destina-se a microcomputadores da linha TRS-80 com BASIC Nível II (para cassete). A rotina de inicialização é assim:

```
1000 ' ROTINA DE INICIALIZACAO
1010 IR=49000
1020 FOR I=1 TO 26
1030 READ N:POKE IR+I,N
1040 NEXT N
1050 DATA 205,127,10,62,1,211
1060 DATA 255,237,95,87,71,16
1070 DATA 254,62,2,211,255,66
1080 DATA 16,254,43,124,181,32
1090 DATA 234,201
1110 POKE 16527,INT(IR/256) :
POKE 16526,IR-INT(IR/256)*256+1
1120 RETURN
```

A rotina de produção de sons:

```
2000 ' SUBROTINA PARA RUIDO
2010 N=DR*1000
2020 I=USR(N):RETURN
```

E um pequeno programa de teste:

```
10 'PROGRAMA DE TESTE
20 GOSUB 1000 :'INICIALIZACAO
30 CLS:INPUT"DURACAO (S)";DR
40 GOSUB 2000:GOTO 30
```

Finalmente, para rodar o programa em micros com Disk BASIC (para disquete), faça as seguintes modificações:

```
1110 DEFUSR0=IR+1
2040 I=USR0(N):RETURN
```

Se você quiser usar as duas rotinas em linguagem de máquina no mesmo programa, altere os endereços de partida IM e IR, de modo a evitar superposição (por exemplo, se IM = 49000, faça IR = 49036, no mínimo).

JOGOS DE GUERRA: O MAPA DA BATALHA

Chegou a hora de desenhar o mapa do campo de batalha. Vamos preenchê-lo com vilas, florestas e montanhas, para, depois, colocar as forças inimigas em suas posições iniciais.

No primeiro artigo desta série sobre jogos de guerra, criamos os blocos gráficos que irão representar as unidades militares no campo de batalha. Esses símbolos serão colocados e movidos em um mapa, para que possamos acompanhar o desenrolar da disputa e planejar nossa estratégia.

O MAPA

Capa e Espada terá uma matriz para representar o mapa. Este permanecerá em exibição durante todo o jogo, ocupando a maior parte do vídeo.

Como o mapa está sempre na tela, poderíamos dispensar a matriz, pois a memória de vídeo conterá todas as informações a respeito do mapa. Convém, no entanto, manter esses dados organizados em uma matriz, mesmo às custas de grande quantidade de memória. Será muito mais fácil obter e modificar as informações na matriz do que na memória de vídeo.

A matriz terá tantos elementos quantas forem as posições do mapa. Os mapas dos micros das linhas MSX, TRS-Color e Spectrum têm trinta por dezesseis posições; os do Apple e do TK-2000, vinte por dezoito.

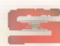
AS TROPAS

Para controlar a posição das tropas no mapa e verificar se há obstáculos em seu caminho quando as movimentamos, precisamos de uma segunda matriz. Esta não só conterá a posição de cada uma das unidades, mas, também, toda e qualquer informação referente a elas.

Em *Capa e Espada*, a matriz da tropa conterá ainda informações relacionadas aos combates, ao movimento etc. O conteúdo de uma matriz desse tipo depende da natureza do jogo. Você terá melhores informações sobre isso à medida que formos ampliando o programa.

DIMENSIONAMENTO DAS MATRIZES

As linhas seguintes dimensionam as matrizes do mapa e da tropa.



```
350 DIM m(16,30)
355 DIM T(16,9)
```



```
350 DIM M(16,30)
355 DIM T(15,8)
```



```
350  DIM M(18,20)
360  DIM T(16,9)
```



```
350 DIM M(16,30)
355 DIM T(16,9)
```

Os valores contidos na matriz do mapa asseguram que ele tenha o tamanho da tela. A matriz da tropa, por sua vez, é dimensionada de modo que possa conter nove tipos de informação a respeito de cada uma das dezesseis unidades que estão em combate.

COMO PREENCHER O MAPA

O próximo passo consiste em determinar os vários tipos de terreno que compõem a área mapeada, bem como a posição inicial de cada uma das unidades. Poderíamos ter optado por um mapa fixo — o que seria muito importante se quiséssemos reproduzir uma batalha famosa. Porém, é bem provável que os jogadores prefiram alterar o campo de batalha a cada jogo. Para fazê-lo sem complicações, utiliza-se o gerador de números aleatórios do micro.

A definição do terreno poderia resumir-se a uma simples escolha ao acaso entre os tipos possíveis. Porém, a distribuição de florestas e montanhas geralmente não é um simples fruto do acaso. Seria interessante que o programa pudesse reproduzir um padrão mais próximo do real, concentrando montanhas ou florestas em certas áreas.

Durante o planejamento do mapa de um jogo de guerra, é importante também considerar o tipo de terreno no qual esperamos que se dê a ação. Em *Capa e Espada*, por exemplo, pretende-se simular uma guerra medieval, com a maioria das batalhas sendo travadas em campo aberto. Nesse caso, não é interessante ter montanhas e florestas demais.

A ESCOLHA DO TERRENO

A rotina que listamos a seguir é essencialmente aleatória, embora haja um certo controle sobre a seleção, o que ajuda a dar uma aparência de maior realidade ao mapa.



```
 20 DEF FN r(x)=INT (RND*x)+1
800 REM Escolhe terreno
810 LET R=FN r(50)
820 IF R>5 THEN  LET R=0
830 IF R>4 THEN  LET R=3:
 RETURN
840 IF R>1 THEN  LET R=2
850 RETURN
```



```
15 R=RND(-TIME)
20 DEF FN R(X)=INT(RND(1)*X)+1
800 REM TERRENO
810 R=FN R(50)
820 IF R>5 THEN R=0
830 IF R>4 THEN R=3:RETURN
840 IF R>1 THEN R=2
850 RETURN
```



```
20  DEF  FN R(X) =  INT ( RND (
1) * X) + 1
800  REM  TERRENO
810 R =  FN R(50)
820  IF R > 5 THEN R = 0
830  IF R > 4 THEN R = 3: RETUR
N
840  IF R > 1 THEN R = 2
850  RETURN
```



```
800 REM   ESCOLHE O TERRENO
```





```
810 R=RND(50)
820 IF R>5 THEN R=0
830 IF R>4 THEN R=3:RETURN
840 IF R>1 THEN R=2
850 RETURN
```

O programa utiliza apenas números aleatórios inteiros. Como o Apple, o TK-2000, o Spectrum e o MSX não têm uma função como a **RND**, a linha 20 **DEF**ine uma Função para isto.

Em todos os programas, um número aleatório inteiro entre 1 e 50 é criado. A rotina de "escolha de terreno" seleciona um valor para a variável **R** de acordo com o número criado. Se esse número for maior que 5, **R** valerá 0, código de campo aberto. Se for 5, **R** será 3, código de montanha; se for 2, 3 ou 4, **R** será 2, valor que representa floresta; e, se for 1, **R** fica valendo 1, código de vila.

CONTROLANDO O ACASO

A rotina de "escolha de terreno" é chamada para definir cada posição da matriz do mapa ao longo de uma de suas dimensões. Como foi mencionado anteriormente, não é desejável que a distribuição seja totalmente aleatória. A rotina que se segue impõe um certo padrão à distribuição dos tipos de terreno, tornando o mapa mais real.

```
370 LET i$="NOSL"
470 REM   Cria
480 FOR i=1 TO 16: GOSUB 800:
LET m(i,1)=R: NEXT i
490 FOR i=1 TO 16
500 FOR j=2 TO 30
510 LET s=FN r(10)
520 IF s<8 THEN  GOSUB 800
525 IF s>=8 THEN  LET R=m(i,j-
1)
530 LET m(i,j)=R
540 IF R=3 AND j<30 THEN  LET
m(i,j+1)=4
550 IF m(i,j)<>0 AND m(i,j)<>3
THEN  PRINT AT i,j;CHR$ (m(i,j
)+143)
555 IF m(i,j)=3 AND j<>30 THEN
 PRINT AT i,j;CHR$ 146;AT i,j+
1;CHR$ 147
560 NEXT j
570 NEXT i
580 GOSUB 720
590 FOR i=1 TO 8
600 FOR j=1 TO 2: LET T(i,j)=2
: LET T(i+8,j)=2: NEXT j
610 FOR j=3 TO 4: READ T(i,j):
LET T(i+8,j)=T(i,j): NEXT j
620 READ mr
630 FOR j=0 TO 8 STEP 8
640 LET T(i+j,5)=mr+FN r(2)
650 LET T(i+j,6)=(FN r(100)*10
)+10
660 LET T(i+j,7)=T(i+j,6)
670 NEXT j
680 LET T(i,8)=15
690 LET T(i+8,8)=1
700 NEXT i
710 RETURN
```



```
370 I$="NWSE"
470 REM CRIAÇAO
480 FOR I=1 TO 16:GOSUB 800:M(I
,1)=R:NEXT I
490 FOR I=1 TO 16
500 FOR J=2 TO 30
510 S=FN R(10)
520 IF S<8 THEN GOSUB 800
525 IF S>=8 THEN R=M(I,J-1)
530 M(I,J)=R
540 IF R=3 AND J<30 THEN M(I,J+
1)=4
550 IF M(I,J)<>0 AND M(I,J)<>3
THEN LOCATE J,I:PRINT CHR$(M(I,
J)+223)
555 IF M(I,J)=3 AND J<> 30 THEN
 LOCATE J,I:PRINT CHR$(226)+CHR
$(227);
560 NEXT J,I
580 GOSUB 720
590 FOR I=1 TO 8
600 FOR J=1 TO 2:T(I,J)=2:T(I+8
,J)=2:NEXT J
610 FOR J=3 TO 4:READ T(I,J):T(
I+8,J)=T(I,J):NEXT J
620 READ MR
630 FOR J=0 TO 8 STEP 8
640 T(I+J,5)=MR+FN R(2)
650 T(I+J,6)=(FN R(100)*10)+10
660 T(I+J,7)=T(I+J,6)
670 NEXT J
680 T(I,8)=15
690 T(I+8,8)=1
700 NEXT I
710 RETURN
```



```
370 I$ = "NOSL"
470  REM  CRIACAO
480  FOR I = 1 TO 18: GOSUB 800
:M(I,1) = R: NEXT
490  FOR I = 1 TO 18
500  FOR J = 1 TO 20
510 S =  FN R(10)
520  IF S < 8 THEN  GOSUB 800
525  IF S >  = 8 THEN R = M(I,J
 - 1)
530 M(I,J) = R
540  IF R = 3 AND J < 20 THEN M
(I,J + 1) = 4
550  IF M(I,J) <  > 0 AND M(I,J
) <  > 3 THEN Y = I:X = J:N = M
(I,J) - 1: GOSUB 10000
560  IF M(I,J) = 3 AND J <  > 2
0 THEN Y = I:X = J:N = 2: GOSUB
 10000:X = J + 1:N = 3: GOSUB 1
0000
570  NEXT J,I
580  GOSUB 720
590  FOR I = 1 TO 8
```

```
600  FOR J = 1 TO 2:T(I,J) = 2:
T(I + 8,J) = 2: NEXT J
610  FOR J = 3 TO 4: READ T(I,J
).T(I + 8,J) = T(I,J): NEXT J
620  READ MR
630  FOR J = 0 TO 8 STEP 8
640 T(I + J,5) = MR +  FN R(2)
650 T(I + J,6) = ( FN R(100) *
10) + 10
660 T(I + J,7) = T(I + J,6)
670  NEXT J
680 T(I,8) = 18
690 T(I + 8,8) = 1
700  NEXT I
710  RETURN
10000 X = X * 2 - 2:N = N * 2:
FOR W = 0 TO 1:X = X + W:N = N
+ W: FOR V = 0 TO 7
10010  POKE T + (Y - 8 * (Y > 7
) - 8 * (Y > 15)) * 128 + 40 *
(Y > 7) + 40 * (Y > 15) + X + 1
024 * V, PEEK (EE + N * 8 + V)
10020  NEXT V,W: RETURN
```



```
370 IS="NOSL"
470 REM CRIAR
480 FOR I=1 TO 16:GOSUB 800·M(I
,1)=R NEXT I
490 FOR I=1 TO 16
500 FOR J=2 TO 30
510 S=RND(10)
520 IF S<8 THEN GOSUB 800
525 IF S>=8 THEN R=M(I,J-1)
530 M(I,J)=R
540 IF R=3 AND J<30 THEN M(I,J+
1)=4
550 IF M(I,J)<>0 AND M(I,J)<>3
THEN LINE (J*8,I*8) (J*8+7,I*8+
7),PRESET,BF:DRAW"BM"+STR$(J*8)
+","+STR$(I*8)+UC$(M(I,J))
555 IF M(I,J)=3 AND J<>30 THEN
DRAW"BM"+STR$(J*8)+","+STR$(I*8
)+UC$(3)+"BM"+STR$((J+1)*8)+","
+STR$(I*8)+UC$(4)
560 NEXT J,I
580 GOSUB 720
590 FOR I=1 TO 8
600 FOR J=1 TO 2·T(I,J)=2:T(I+8
,J)=2 NEXT J
610 FOR J=3 TO 4:READ T(I,J):T(
I+8,J)=T(I,J) NEXT J
620 READ MR
630 FOR J=0 TO 8 STEP 8
640 T(I+J,5)=MR+RND(2)
650 T(I+J,6)=(RND(100)*10)+10
660 T(I+J,7)=T(I+J,6)
670 NEXT J
680 T(I,8)=15
690 T(I+8,8)=1
700 NEXT I
710 RETURN
```

A rotina gera um novo número aleatório, S, para cada elemento restante da matriz. O valor de S, selecionado na linha 510, varia de 1 a 10. A linha 520 faz com que, em 70% das vezes, o novo bloco de terreno seja escolhido ao acaso — se S<8, o programa chama a sub-rotina de escolha de terreno. Os 30% restantes serão constituídos de blocos iguais aos vizinhos da esquerda. Esse procedimento tem como objetivo criar grupos de blocos no mapa. As linhas que vão de 550 a 570 colocam os blocos na tela.

POSICIONAMENTO DAS TROPAS

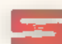
As posições dos combatentes são armazenadas nas matrizes das tropas como pares de coordenadas — horizontais e verticais.

Inicialmente, os adversários ficam em extremos opostos da tela. No campo de *Capa e Espada*, o jogador começa no extremo sul (margem inferior do mapa), e o computador, no norte (topo da tela).

Portanto, a coordenada vertical já está previamente determinada.

A coordenada horizontal precisa ser selecionada. Assim como para a definição do tipo de terreno, é interessante que haja um elemento de acaso nessa escolha. Contudo, algumas restrições devem ser feitas — precisamos impedir, por exemplo, que duas unidades ocupem o mesmo espaço.

A rotina que listamos a seguir seleciona a posição inicial de cada unidade, de ambos os lados. Em primeiro lugar, as tropas são selecionadas ao acaso. Depois, o mapa é dividido verticalmente em oito colunas. Cada coluna representa os limites dentro dos quais as unidades serão colocadas. A posição definitiva de todas elas é, então, selecionada de acordo com os limites da coluna.



```
860 REM Dispoe tropas
870 INK 2
880 FOR m=1 TO 2
890 LET s=1: LET r=1
900 FOR k=1 TO 8
910 REM  Loop
920 LET s=FN r(8*m)
930 IF T(s,9)<>0 THEN  GOTO
 910
940 LET r=FN r(4)+r
950 LET r=r-INT (r/30)
960 LET T(s,9)=r
970 INK m
980 PRINT AT T(s,8),T(s,9);u$(
s)
 990 NEXT k
1000 NEXT m
1010 RETURN
```



```
860 REM TROPAS
880 FOR M=1 TO 2
890 S=1:R=1
900 FOR K=1 TO 8
920 S=FN R(8*M)
930 IF T(S,9)<>0 THEN 920
940 R=FN R(4)+R
950 R=R-INT(R/30)
960 T(S,9)=R
980 LOCATE T(S,9),T(S,8):PRINT
CHR$(U(S))
990 NEXT K,M
1010 RETURN
```



```
860  REM  DISTRIBUICAO
880  FOR M = 1 TO 2
890 S = 1:R = 1
900  FOR K = 1 TO 8
920 S =  FN R(8 * M)
930  IF T(S,9) <  > 0 THEN 920
940 R =  FN R(3) + R
950 R = R -  INT (R / 20)
960 T(S,9) = R
980 Y = T(S,8):X = T(S,9):N =
VAL ( MID$ (U$,S - (M - 1) * 8,
1)) + (M - 1) * 5: GOSUB 10000
990  NEXT K,M
1010  RETURN
```



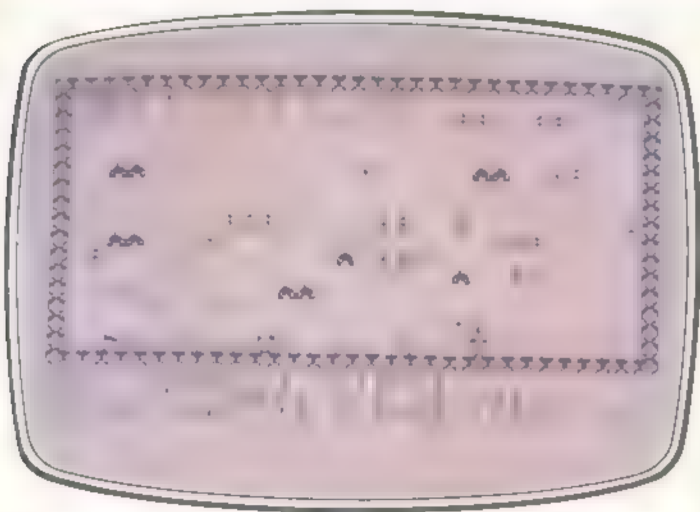
```
860 REM  DISPOE TROPAS
870 COLOR 2
880 FOR M=1 TO 2
890 S=1:R=1
900 FOR K=1 TO 8
910 REM
920 S=RND(8*M)
930 IF T(S,9)<>0 THEN 910
940 R=RND(4)+R
950 R=R-INT(R/30)
960 T(S,9)=R
970 COLOR M:IF M=1 THEN COLOR 3
980 DRAW"BM"+STR$(T(S,9)*8)+","
+STR$(T(S,8)*8):UU=VAL(MID$(U$,
S,1)):A$=UC$(UU):GOSUB 3000
990 NEXT K
1000 NEXT M
1010 RETURN
```

Como as oito unidades são escolhidas ao acaso, a posição inicial de cada uma varia e, assim, cada novo jogo é diferente do anterior.

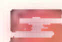
Os dois exércitos estão armazenados em uma só matriz. Por isso, a mesma rotina pode ser usada para escolher a posição inicial de todas as tropas, bastando, apenas, um laço **FOR... NEXT** (linhas 880 e 1000). O laço também faz com que os dois exércitos apareçam em cores diferentes.

ÀS ARMAS

Uma vez determinadas as posições iniciais das unidades, elas devem ser colocadas no vídeo.



Capa e Espada: **mapa do campo de batalha no Spectrum.**



```
410 LET U$=CHR$ 148+CHR$ 149+
CHR$ 150+CHR$ 150+CHR$ 151+
CHR$ 151+CHR$ 152+CHR$ 152:
LET U$=U$+U$
```



```
410 U(1)=228:U(2)=229:U(3)=230:
U(4)=230:U(5)=231:U(6)=231:U(7)
=232:U(8)=232
```



```
410 U$ = "45667788"
```



```
410 U$="65778899":U$=U$+U$
```

Na linha 410 definimos um cordão U$ para armazenar os códigos dos caracteres que representam cada unidade. No caso do MSX, utilizamos uma variável indexada U( ).

MOLDURA

A tela fica mais bonita — e menos confusa — se fizermos uma moldura para o campo de batalha.



```
720 REM   Borda Decorativa
730 FOR i=0 TO 16
740 PRINT AT i,0;CHR$ 150;AT i
,31;CHR$ 150
750 NEXT i
760 FOR i=0 TO 31
770 PRINT AT 0,i;CHR$ 150;AT
16,i;CHR$ 150
780 NEXT i
790 RETURN
```



```
720 REM BORDA
730 FOR I=0 TO 31
740 VPOKE BASE(5)+I,246
750 VPOKE BASE(5)+I+544,246
760 NEXT I:FOR I=0 TO 574 STEP
32
770 VPOKE BASE(5)+I,246
780 VPOKE BASE(5)+I-1,246
790 NEXT I:RETURN
```



```
720  REM  BORDA
730  FOR YY = 0 TO 19 STEP 19
740  FOR XX = 1 TO 20
750 N = 11:X = XX:Y = YY
760 X = X * 2 - 2:N = N * 2: FO
R W = 0 TO 1:X = X + W:N = N +
W: FOR V = 0 TO 7
770  POKE T + (Y - 8 * (Y > 7)
- 8 * (Y > 15)) * 128 + 40 * (Y
 > 7) + 40 * (Y > 15) + X + 102
4 * V, PEEK (E + N * 8 + V) - 1
28
780  NEXT V,W
790  NEXT XX,YY: RETURN
```



```
720 REM BORDA
730 LINE (0,128)-(255,135),PRES
ET,BF
740 LINE(248,0)-(255,191),PRESE
T,BF:LINE(4,4)-(252,132),PSET,B
790 RETURN
```

Essa rotina simplesmente desenha, repetidas vezes, em torno do mapa, o símbolo utilizado para os lanceiros, só que em outra cor.

MOBILIZAÇÃO DAS FORÇAS

Agora que o mapa está preparado, veremos como os jogadores movem suas unidades. Toda a mobilização decorre de ordens — explicadas no próximo artigo. Porém, antes que se possa completar o movimento, é preciso definir vários detalhes do jogo:

- Distância máxima, em número de posições, que cada unidade pode atingir em um só movimento. Em *Capa e Espada*, a mobilidade de uma unidade depende apenas do peso de sua armadura, mas, em outros jogos, fatores como moral, disciplina e cansaço, entre outros, podem ser levados em conta.

- Existência de vantagens e bônus. Em nosso programa, os bônus são dados apenas aos cavaleiros. Porém, é possível destiná-los também a unidades que realizem determinadas ações — como subir uma ladeira ou transportar carga — ou, ainda, que possuam um comandante muito habilidoso.

- Obstáculos ao movimento. O programador deve decidir que tipo de influência o terreno exerce sobre o movimento de tropas. Em nosso caso, todos os terrenos, menos o campo aberto, diminuem a distância máxima em uma unidade. Também é preciso verificar se há outra unidade no caminho.

- A borda do mapa foi alcançada?

Adicione mais esta rotina e o programa poderá movimentar as tropas levando em conta todos esses fatores.



```
1160 REM   Move unidade
1170 LET ox=T(b,8): LET oy=t(b,
9)
1175 LET z$=""
1180 IF m(T(b,8),T(b,9))<>0 THE
N  LET z$=CHR$ (143+m(T(b,8),T(
b,9)))
1190 LET D=5-T(b,4)
1200 IF b<3 OR b=9 OR b=10 THEN
 LET D=D+2
1210 LET v=T(b,2)-1
1215 LET up=0: LET al=v-2
```

```
1220 IF v/2-(INT (v/2))=0 THEN
 LET up=v-1: LET al=0
1230 REM  Repeticao
1240 LET nl=T(b,9)+al· LET np=T
(b,8)+up
1250 IF np<1 THEN  LET np=1
1260 IF np>15 THEN  LET np=15
1270 IF nl<1 THEN  LET nl=1
1280 IF nl>30 THEN  LET nl=30
1290 IF m(np,nl)>0 THEN  LET D=
D-1
1300 FOR k=1 TO 8
1310 IF (T(k,9)=nl AND T(k,8)=n
p AND k<>b) THEN  LET D=0
1315 IF (T(k+8,9)=nl AND T(k+8,
8)=np AND k+8<>b) THEN  LET D=0
1320 NEXT k
1330 IF d>0 THEN  LET T(b,9)=nl
: LET T(b,8)=np: LET D=D-1
1340 IF D<>0 THEN  GOTO 1230
1350 INK 0: PRINT AT ox,oy;z$
1360 INK cl: PRINT AT T(b,8),T(
b,9);u$(1)
1370 RETURN
```



```
1160 REM MOVE
1170 OX=T(B,8):OY=T(B,9)
1175 GH=32
1180 IF M(T(B,8),T(B,9)<>0 THEN
 GH=223+M(T(B,8).T(B,9))
1190 D=5-T(B,4)
1200 IF B<3 OR B=9 OR B=10 THEN
 D=D+2
1210 V=T(B,2)-1
1215 UP=0:AL=V-2
1220 IF V/2-(INT(V/2))=0 THEN U
P=V-1:AL=0
1240 N1=T(B,9)+AL:NP=T(B,8)+UP
1250 IF NP<1 THEN NP=1
1260 IF NP>15 THEN NP=15
1270 IF N1<1 THEN N1=1
1280 IF N1>30 THEN N1=30
1290 IF M(NP,N1)>0 THEN D=D-1
1300 FOR K=1 TO 8
1310 IF (T(K,9)=N1 AND T(K,8)=N
P AND K<>B) THEN D=0
1315 IF (T(K+8,9)=N1 AND T(K+8,
8)=NP AND K+8<>B) THEN D=0
1320 NEXT K
1330 IF D>0 THEN T(B,9)=N1:T(B,
8)=NP:D=D-1
1340 IF D<>0 THEN 1240
1350 LOCATE OY,OX:PRINT CHR$(GH
);
1360 LOCATE T(B,9),T(B,8):PRINT
 CHR$(U(I)+CL)
1370 RETURN
```



```
1160  REM  MOVE
1170 OX = T(B,8):OY = T(B,9)
1175 GH = 14
1180  IF M(T(B,8),T(B,9)) <  >
0 THEN GH = M(T(B,8),T(B,9)) -
1
1190 D = 5 - T(B,4)
1200  IF B < 3 OR B = 9 OR B =
10 THEN D = D + 2
1210 V = T(B,2) - 1
1215 UP = 0:AL = V - 2
1220  IF V / 2 - ( INT (V / 2))
 = 0 THEN UP = V - 1:AL = 0
1240 N1 = T(B,9) + AL:NP = T(B,
8) + UP
1250  IF NP < 1 THEN NP = 1
1260  IF NP > 18 THEN NP = 18
1270  IF N1 < 0 THEN N1 = 1
1280  IF N1 > 19 THEN N1 = 20
1290  IF M(NP,N1) > 0 THEN D =
D - 1
1300  FOR K = 1 TO 8
1310  IF (T(K,9) = N1 AND T(K,8
) = NP AND K <  > B) THEN D = 0
1315  IF (T(K + 8,9) = N1 AND T
(K + 8,8) = NP AND K + 8 <  > B
) THEN D = 0
1320  NEXT K
1330  IF D > 0 THEN T(B,9) = N1
:T(B,8) = NP:D = D - 1
1340  IF D <  > 0 THEN 1240
1350 X = OY:Y = OX:N = GH: GOSU
B 10000
1360 X = T(B,9):Y = T(B,8):N =
 VAL ( MID$ (U$,I - (I > 8) * 8
,1)) + (I > 8) * 5: GOSUB 10000
1370  RETURN
```



```
1160 REM  MOVE UNIDADE
1170 OX=T(B,8):OY=T(B,9)
1175 ZZ=0
1180 IF M(T(B,8),T(B,9))<>0 THE
N ZZ=M(T(B,8),T(B,9))
1190 D=5-T(B,4)
1200 IF B<3 OR B=9 OR B=10 THEN
 D=D+2
1210 V=T(B,2)-1
1215 UP=0:AL=V-2
1220 IF (V/2)-INT(V/2)=0 THEN U
P=V-1:AL=0
1230 REM
1240 NL=T(B,9)+AL:NP=T(B,8)+UP
1250 IF NP<1 THEN NP=1
1260 IF NP>15 THEN NP=15
1270 IF NL<1 THEN NL=1
1280 IF N>30 THEN NL=30
1290 IF M(NP,NL)>0 THEN D=D-1
1300 FOR K=1 TO 8
1310 IF (T(K,9)=NL AND T(K,8)=N
P AND K<>B) THEN D=0
1315 IF (T(K+8,9)=NL AND T(K+8,
8)=NP AND K+8<>B) THEN D=0
1320 NEXT K
1330 IF D>0 THEN T(B,9)=NL:T(B,
8)=NP:D=D-1
1340 IF D<>0 THEN 1230
1350 X9=OY*8:Y9=OX*8:IF ZZ<>0 T
HEN COLOR 4:LINE(X9,Y9)-(X9+7,Y
9+7),PRESET,BF:DRAW"BM"+STR$(X9
)+","+STR$(Y9)+UC$(ZZ) ELSE LIN
E (X9,Y9)-(X9+7,Y9+7),PRESET,BF
1360 COLOR CL:DRAW"BM"+STR$(T(B
,9)*8)+","+STR$(T(B,8)*8):UU=VA
L(MID$(U$,I,1)):A$=UC$(UU):GOSU
B 3000
1370 RETURN
```

Os testes vão aumentar, reduzir ou impedir completamente a mobilidade das tropas. A rotina primeiro "lembra-se" da posição anterior e do tipo de terreno que havia naquela posição. Em seguida, calcula a direção e o deslocamento máximo.

As linhas 1230 a 1340 formam um laço que testa cada posição ao longo da trajetória da unidade, verificando se são ocupadas por tropas ou tipos de terreno que afetam o movimento. De acordo com o que esse laço encontrar, o deslocamento final é calculado. O laço se repete até que a distância ao ponto de destino seja zero.

Após decidir sobre o deslocamento, a rotina coloca o símbolo do terreno original na posição anterior e desenha a tropa na nova posição.

As listagens aqui apresentadas ainda não poderão ser completamente testadas. Como está, o programa apenas desenha o mapa, sendo interrompido por uma mensagem de "falta de dados". Com as rotinas do próximo artigo, que trata das ordens dadas pelo jogador, esse problema deixará de existir.



O que é uma simulação aleatória?

Os jogos de simulação geralmente envolvem uma metodologia de cálculo específica para a área que está sendo simulada. Na maioria das simulações isso abrange tanto fórmulas matemáticas quanto regras de decisão (regras heurísticas).

Entretanto, o jogo de simulação fica muito previsível se não contiver algum elemento de probabilidade (ou seja, um sorteio ao acaso realizado para uma equação matemática para uma regra de decisão). As situações passam a se repetir exatamente da mesma maneira frente a um determinado conjunto de condições.

Por isso, é importante usar o gerador interno de números aleatórios do computador (funções **RAND**, **RND** etc.) para sortear ao acaso o caminho a ser seguido ou o valor numérico de algum dado de simulação.

São exemplos do que pode ser gerado em programas de simulação:

- O nome de uma nave espacial e o de seu comandante.
- O número e a direção das saídas de uma câmara secreta em um jogo de aventuras.
- A permanência do jogador na prisão, no Jogo Imobiliário.

JOGOS DE GUERRA: A ARTE DE COMANDAR

- O INÍCIO DA PARTIDA
- COMO DAR ORDENS À TROPA
- MOVIMENTO E DIREÇÃO
- STATUS
- O COMPUTADOR COMO JOGADOR



Mobilize seu exército para a guerra, definindo uma estratégia de ação e disciplinando a tropa. Esses requisitos são fundamentais para o exercício da arte de comandar.

Agora que as rotinas de posicionamento e movimentação das tropas já foram incorporadas ao programa, é preciso começar a dar ordens ao exército.

Diversos fatores tendem a influenciar o comportamento de cada unidade no campo de batalha. Os mais significativos dentre eles são os seguintes:

- Última ordem recebida pela unidade.
- Direção em que a unidade se move.
- Tipo de munição utilizado.
- Armaduras e demais equipamentos de que dispõe a unidade.
- Poder destrutivo inicial.
- Poder destrutivo no momento em que a batalha é travada.
- Moral da tropa.
- Posição estratégica assumida pela unidade.
- Terreno em que a batalha tem lugar.

Esses fatores correspondem aos nove elementos da matriz da tropa, que dimensionamos no artigo anterior. O terreno e a posição já foram definidos por ocasião da rotina de movimento. Neste artigo atribuiremos valores aos elementos restantes.

Ao começar o jogo, os valores iniciais adequados são colocados na matriz da tropa. Os tipos de armadura e de munição serão sempre os mesmos. Da mesma forma, o moral será também quase sempre o mesmo, embora possa sofrer algumas variações — os camponeses provavelmente nunca estarão muito dispostos a lutar (afinal, a guerra não é deles, mas dos barões feudais), enquanto os cavaleiros, para manter as aparências, nunca agirão covardemente. A capacidade destrutiva poderá ser muito diferente de uma partida para a outra. As ordens e as direções iniciais são determinadas pelo programa: todos começam parados ("ALTO"), de modo que não existe uma direção definida.

As rotinas que se seguem acertam o valor de diversas variáveis. Os dados são obtidos de linhas **DATA**, ou através de alguns cálculos, ou das duas maneiras. Quando o programa estiver completo e em funcionamento, poderemos modificar esses dados de acordo com nossas preferências.

ABANDONE O QUARTEL-GENERAL

Estas rotinas acertam os valores iniciais dos elementos que ainda não foram definidos:

S

```
190 REM Inicializacao
200 LET vc=0: LET de=0
310 REM
320 PAPER 7
330 INK 0
340 CLS
360 DIM t$(8,12): DIM o$(5,12)
: DIM w$(5,9): DIM m$(5,12):
DIM a$(4,12): DIM r$(4,12):
DIM c(8)
380 FOR i=1 TO 5: READ o$(i),
w$(i),m$(i): NEXT i
390 FOR i=1 TO 8: READ t$(i):
NEXT i
400 FOR i=1 TO 4: READ a$(i),
r$(i): NEXT i
410 LET u$=CHR$ 148+CHR$ 149+
CHR$ 150+CHR$ 151+CHR$ 151+
CHR$ 152+CHR$ 152: LET u$=u$+
u$
415 LET x$="NnSs"
420 RETURN
2760 DATA "fogo","nada","covard
e","alto","arco","baixo"
2770 DATA "marche","espada","di
sposto","status","machado","bra
vo"
2780 DATA "retirada","lanca","v
alente"
2790 DATA "cavaleiros","sargent
os","lanceiros","arqueiros","ar
queiros","camponeses","campones
es"
2800 DATA "nada","campo","gibao
","vila","malha metalica","flor
esta","chapa metalica","montanh
a"
2810 DATA 5,4,3,5,3,3,4,3,2,3,3
,1,2,2,1,2,3,2,3,2,0,3,1,0
```

MSX

```
190 REM INICIO
200 VC=0:DE=0
340 CLS
360 DIM T$(8),O$(5),W$(5),M$(5)
,A$(4),R$(4)
380 FOR J=1 TO 5:READ O$(J),W$(
J),M$(J):NEXT J
390 FOR J=1 TO 8:READ T$(J):NEX
T J
400 FOR J=1 TO 4:READ A$(J),R$(
J):NEXT J
415 X$="NnSs"
420 RETURN
2760 DATA FOGO,NADA,COVARDE,ALT
O,ARCO,BAIXO
2770 DATA MARCHE,ESPADA,DISPOST
O,STATUS,MACHADO,BRAVO
2780 DATA RETIRADA,LANCA,VALENT
E
2790 DATA CAVALEIROS,SARGENTOS,
LANCEIROS,LANCEIROS,ARQUEIROS,A
RQUEIROS,CAMPONESES,CAMPONESES
2800 DATA NADA,CAMPO,GIBÃO,VILA
,MALHA METALICA,FLORESTA,PLACA
METALICA,MONTANHA
2810 DATA 5,4,3,5,3,3,4,3,2,3,3
,1,2,2,1,2,3,2,3,2,0,3,1,0
```





```
190  REM  INICIO
200 CL = 0:VC = 0:DE = 0
340 HGR2 : POKE - 16301,0: GO
SUB 2540
360  DIM T$(8),O$(5),W$(5),M$(5
),A$(4),R$(4)
380  FOR I = 1 TO 5: READ O$(I)
,W$(I),M$(I): NEXT I
390  FOR I = 1 TO 8: READ T$(I)
: NEXT I
400  FOR I = 1 TO 4: READ A$(I)
,R$(I): NEXT I
410 U$ = "45667788"
415 X$ = "NNSS"
420  RETURN
2760  DATA  FOGO,NADA,COVARDE,A
LTO,ARCO,BAIXO,MARCHE,ESPADA,DI
SPOSTO
2780  DATA  STATUS,MACHADO,BRAV
O,RETIRADA,LANCA,VALENTE
2790  DATA  CAVALEIROS,SARGENTO
S,LANCEIROS,LANCEIROS,ARQUEIROS
,ARQUEIROS,CAMPONESES,CAMPONESE
S
2800  DATA  NADA,CAMPO,GIBAO,VI
LA,MALHA METALICA,FLORESTA,CHAP
A METALICA,MONTANHA
2810  DATA   5,4,3,5,3,3,4,3,2,
3,3,1,2,2,1,2,3,2,3,2,0,3,1,0
```



Os usuários do micro TK-2000 devem modificar as seguintes linhas do programa destinado ao Apple.

```
340  HGR2   GOSUB 2540
```



```
310 REM
330 COLOR 4,2
340 PCLS
360 DIM T$(8),O$(5),W$(5),M$(5)
,A$(4),R$(4)
380 FOR J=1 TO 5:READ O$(J),W$(
J),M$(J):NEXT J
390 FOR J=1 TO 8:READ T$(J):NEX
T J
400 FOR J=1 TO 4:READ A$(J),R$(
J):NEXT
415 X$="NnSs"
420 RETURN
2760 DATA FOGO,NADA,COVARDE,ALT
O,ARCO,BAIXO
2770 DATA MARCHE,ESPADA,DISPOST
O,STATUS,MACHADO,BRAVO
2780 DATA RETIRADA,LANCA,VALENT
E
2790 DATA CAVALEIROS,SARGENTOS,
LANCEIROS,LANCEIROS,ARQUEIROS,A
RQUEIROS,CAMPONESES,CAMPONESES
2800 DATA NADA,CAMPO,GIBAO,VILA
,MALHA METALICA,FLORESTA,CHAPA
METALICA,MONTANHA
2810 DATA 5,4,3,5,3,3,4,3,2,3,3
,1,2,2,1,2,3,2,3,2,0,3,1,0
```

Os valores iniciais são obtidos com **READ** das linhas **DATA**, juntamente com as palavras equivalentes aos valores numéricos colocados na matriz. As palavras serão usadas na janela de texto para que a batalha se desenrole de maneira clara para o jogador.

DÊ AS ORDENS

O jogo consiste basicamente em dar ordens às unidades — sem isto não haverá combate e, consequentemente, vitória ou derrota.

Existem quatro ordens básicas no jogo *Capa e Espada*: "Fogo", "Alto", "Marche" e "Status". A ordem de atirar se aplica somente aos arqueiros, que obedecerão mesmo que não haja unidades inimigas nas redondezas.

Dar ordens às tropas tem suas desvantagens: quanto mais real quisermos que seja um jogo de guerra, mais complicadas serão essas ordens. Inversamente, quanto mais simples for o jogo, mais fácil será o processo de comando. Assim, um jogo muito realista será mais difícil de jogar; e, se quisermos produzir um jogo fácil de jogar, teremos que sacrificar os detalhes. Depois de programar alguns jogos desse tipo, você será capaz de criar programas mais complexos.

ROTINA DE COMANDO

A rotina de comando de *Capa e Espada* seleciona as unidades de cada jogador, uma a uma, para que estes possam dar as ordens. A unidade em questão tem sua cor modificada no mapa (em alguns computadores a mudança é sutil). Uma mensagem solicitando as novas ordens para essa unidade (indicada por seu número) é mostrada na janela de texto. Se o jogador não quiser alterar as ordens, basta pressionar a tecla N. Se S for pressionada, será mostrado um menu com as ordens possíveis.

Caso seja selecionada a ordem de atirar (**FOGO**) e a unidade não seja constituída de arqueiros, uma nova ordem será escolhida. O comando **ALTO** faz a unidade permanecer onde está e **MARCHE** exige a definição da direção, indicada pelas iniciais de norte, sul, leste e oeste. O programa não faz nenhum teste para verificar se a direção é válida; portanto, preste atenção ao mapa.

COMO SELECIONAR UMA UNIDADE

Quando chega a vez do jogador, a primeira unidade é destacada por meio da mudança de cor; isso se repete com cada uma das outras sete unidades. O símbolo que muda de cor é a unidade que o jogador deve considerar: as ordens devem ou não ser modificadas? Quando a unidade é destacada, uma mensagem surge na janela de texto, mostrando a última ordem dada.



```
1380 REM unidade
1390 GOSUB 2540
1400 INK 0
1410 PRINT  FLASH 1;AT T(i,8),T
(i,9);u$(i)
1420 PRINT AT 17,0;"Unidade num
ero ";i;" ";t$(i);"        "
1430 PRINT AT 18,0;"Ultima orde
m:";o$(T(i,1));" ".
1440 IF T(i,1)=3 THEN  PRINT AT
 18,28;i$(T(i,2))
1450 REM Loop
1460 PRINT AT 19,0;"Deseja muda
```

```
r as ordens (S/N)?"
1465 LET y=0· LET y$=INKEY$
1466 IF y$="" THEN  GOTO 1465
1470 FOR k=1 TO 4
1475 IF x$(k)=y$ THEN  LET y=k
1480 NEXT k
1490 IF y=0 THEN  GOTO 1450
1500 RETURN
```



```
1380 REM UNIDADE
1390 GOSUB 2540
1410 LOCATE T(I,9),T(I,8):PRINT
 CHR$(U(I)+48);
1420 LOCATE 0,18:PRINT "UNIDADE
";I;":";T$(I);"          "
1430 LOCATE 0,19:PRINT "ULTIMA
ORDEM·";O$(T(I,1));
1440 IF T(I,1)=3 THEN PRINT I$(
T(I,2))
1460 LOCATE 0,20:PRINT "QUER MU
DAR AS ORDENS (S/N)?"
1465 YW=0.Y$=INKEY$
1466 IF Y$="" THEN 1465
1470 YW=INSTR(1,X$,Y$)
1490 IF YW=0 THEN 1460
1500 RETURN
```



```
1380  REM  UNIDADE
1390  GOSUB 2540
1410 X = T(I,9)·Y = T(I,8)·N =
 VAL ( MID$ (U$,I - (I > 8) * 8
,1)) + (I > 8) * 5  GOSUB 20000
1420  VTAB 21· PRINT "UNIDADE N
UMERO ";I;" ";T$(I)
1430  PRINT "ULTIMA ORDEM· ";O$
(T(I,1));" ";
1440  IF T(I,1) = 3 THEN  PRINT
  MID$ (I$,T(I,2),1): GOTO 1460
1450  PRINT
1460  PRINT "DESEJA MUDAR AS OR
DENS (S/N)?": GOSUB 30000
1465 YW = 0: GET Y$
1470  FOR K = 1 TO 4
1475  IF  MID$ (X$,K,1) = Y$ TH
EN YW = K
1480  NEXT K
1490  IF YW = 0 THEN 1450
1500  RETURN
20000 X = X * 2 - 2:N = N * 2:
FOR W = 0 TO 1:X = X + W:N = N
+ W· FOR V = 0 TO 7
20010  POKE T + (Y - 8 * (Y > 7
) - 8 * (Y > 15)) * 128 + 40 *
(Y > 7) + 40 * (Y > 15) + X + 1
024 * V, PEEK (EE + N * 8 + V)
- 128
20020  NEXT V,W: RETURN
```



Faça a seguinte modificação no programa do Apple:

```
1460  PRINT "DESEJA MUDAR AS OR
DENS (S/N)?"
```



```
1380 REM UNIDADE
1390 GOSUB 2540
1400 COLOR 4
1410 DRAW"BM"+STR$(T(I,9)*8)+",
"+STR$(T(I,8)*8):UU=VAL(MID$(U$
,I,1)).A$=UC$(UU):GOSUB 3030
1420 DRAW"BM0,144":A$="UNIDADE"
+SR$(I)+" "+T$(I)+"          ":GOS
UB 3190
1430 DRAW"BM0,152":A$="ORDENS S
AO PARA   "+O$(T(I,1))+" ":GOSU
B 3190
1440 IF T(I,1)=3 THEN DRAW "BM1
60,152":A$=MID$(I$,T(I,2),1):GO
SUB 3190
1450 REM
1460 DRAW"BM0,160":A$="DESEJA M
UDAR AS ORDENS (S/N)":GOSUB 319
0
1465 Y=0:Y$=INKEY$
1466 IF Y$="" THEN 1465
1470 Y=INSTR(1,X$,Y$)
1490 IF Y=0 THEN 1450
1500 RETURN
3000 X9=PEEK(200):Y9=PEEK(202):
LINE(X9,Y9)-(X9+7,Y9+7),PRESET,
BF
3010 POKE 200,X9:POKE 202,Y9:DR
AW A$
3020 RETURN
3030 X9=PEEK(200):Y9=PEEK(202):
C9=PEEK(178).COLOR 2 LINE(X9,Y9
)-(X9+7,Y9+7),PSET,BF
3040 POKE 178,C9:GOTO 3010
3050 REM DADOS PARA LERAS E DIG
ITOS
3060 DATA BR4,BDD5RU3NR2U2ERFND
5BR4BU,D6R3EUHEUHBR5,NR5D6R5BR3
BU6,NR3D6R3EU4BUBR4,NR5D3NR3D3R
5BR3BU6,NR5D3NR3D3BR8U6,NR5D6R
5U2BU4BR3
3070 DATA ND6D3R5D3U6BR3,R5L2D6
L3R5BR3BU6,BD5RFREU5L2BR6,ND6D3
R3FD2BU4U2BR4,D6R5BR3BU6,ND6DR5
NL5UBR3,ND6DR2D3R3D2U6BR,
3080 DATA D6R5U6NL5BR3,D6U3R5U3
NL5BR3,D6UR3FRBU2U4NL5BR3,D6U2R
3FDUBU2NL2U3NL4BR4,NR5D3R5D3NL5
BU6BR3,R5L2ND6BR5,D6R5U6BR3
3090 DATA D4RFDRUEU4BR4,D6UR5DU
6BR3,D2RFGND2ERFND2HEU2BR4,D2RF
D3RU3EU2BR4,R4D2GLGD2R4BU6BR3
3100 DATA BR3LGD4FREU4BR4BU,BR2
DNL2D5L2R5BU6BR3,BDRERFDGL2D3R4
BU6BR3,BDRERFDGFDGLHLBU5BR8,D4R
5UD3BU6BR3,NR5D2R3FD2GLHLBU5BR8
,BDBR5LHLGD2NR2D2FREUBU4BR4
3110 DATA R5D2LGLGD2BR7BU6,BR3L
GDFGDFREUHEUBUBR4,BR3LGDFR2D2GL
HLBU2BR4U2BUBR4
3120 REM MATRIZ DE CARACTERES
3130 DIM LE$(26)
3140 FOR K9=0 TO 26:READ LE$(K9
):NEXT
3150 FOR K9=0 TO 9·READ NU$(K9)
:NEXT
3170 RETURN
3180 REM ROTINA PARA IMPRIMIR A
$
3190 FOR K9=1 TO LEN(A$)
3200 B$=MID$(A$,K9,1)
3210 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 3240
3220 IF B$=" " THEN N9=0 ELSE N
9=ASC(B$)-64
3230 DRAW LE$(N9)
3240 NEXT
3250 RETURN
```

As ordens para a unidade destacada são mostradas na janela de texto. O jogador é indagado, então, sobre se deseja mudar as ordens.

O micro TRS-Color possui linhas adicionais (3000 a 3250) destinadas a desenhar as letras e os números na tela de alta resolução.

A ARTE DE COMANDAR

Esta rotina mostra um menu com as ordens possíveis:



```
1900 REM  acao
1910 GOSUB 2540
1920 PRINT AT 18,0;"Opcoes  "
1930 FOR j 1 TO 4
1940 PRINT AT 17+j,8;o$(j,1);"-
",o$(j)
1950 NEXT j
1960 REM loop
1962 LET a 0
1965 LET f$="FfAaMmSs"
1970 LET g$ INKEY$: IF g$="" TH
EN  GOTO 1970
1975 FOR k=1 TO 8
1980 IF f$(k)=g$ THEN  LET a=IN
T ((k+1)/2)
1985 NEXT k
1990 IF a<=0 THEN  GOTO 1960
2000 IF i<>6 AND i<>5 AND a=1 T
HEN  GOSUB 2540: PRINT AT 18,8;
"Nao ha arcos"· GOSUB 2410· GOT
O 1910
2010 IF a=4 THEN  GOSUB 2440: R
ETURN
2020 LET T(i,1)=a
2030 IF a=3 THEN  GOSUB 2050
2040 RETURN
```



```
1900 REM AÇÃO
1910 GOSUB 2540
1920 LOCATE 0,19·PRINT "OPÇÕES
";
1930 FOR J=1 TO 4
1940 LOCATE 8,18+J:PRINT LEFT$(
O$,2);"-";O$(J)
1950 NEXT J
1960 A=0
1965 F$="FHMS"
1970 G$=INKEY$:IF G$="" THEN 19
70
1980 A=INSTR(1,F$,G$)
```

```
1990 IF A<=0 THEN 1960
2000 IF I<>6 AND I<>5 AND A=1 T
HEN GOSUB 2540:LOCATE 8,19:PRIN
T "SEM ARCOS":GOSUB 2410:GOTO 1
910
2010 IF A=4 THEN GOSUB 2440:RET
URN
2020 T(I,1)=A
2030 IF A=3 THEN GOSUB 2050
2040 RETURN
```



```
1900  REM  ACAO
1910  GOSUB 2540
1920  VTAB 21: PRINT "OPCOES: "
;
1930  FOR J = 1 TO 4
1940  HTAB 10: PRINT  LEFT$ (O$
(J),1);" - ",O$(J);: IF J < 4 T
HEN  PRINT " "
1950  NEXT J: GOSUB 30000
1960 F$ = "FFAAMMSS"
1970  GET G$
1973 A = 0
1975  FOR K = 1 TO 8
1980  IF  MID$ (F$,K,1) = G$ TH
EN A =  INT ((K + 1) / 2)
1985  NEXT K
1990  IF A <  = 0 THEN 1970
2000  IF I <  > 6 AND I <  > 5
AND A = 1 THEN  GOSUB 2540: PRI
NT "NAO HA ARCOS": GOSUB 30000:
 GOSUB 2410: GOTO 1910
2010  IF A = 4 THEN  GOSUB 2440
: RETURN
2020 T(I,1) = A
2030  IF A = 3 THEN  GOSUB 2050
2040  RETURN
```



Modificações para o TK-2000:

```
1950  NEXT J
2000  IF I <  > 6 AND I <  > 5
AND A = 1 THEN  GOSUB 2540: PRI
NT "NAO HA ARCOS": GOSUB 2410:
GOTO 1910
```



```
1900 REM SELECIONA ACAO
1910 GOSUB 2540
1920 DRAW"BM0,152":A$="OPCOES
   ":GOSUB 3190
1930 FOR J=1 TO 4
1940 DRAW"BM104,"+STR$(144+J*8)
```

```
:A$=LEFT$(O$(J),1)+"  "+O$(J):G
OSUB 3190
1950 NEXT J
1960 REM
1962 A=0
1965 F$="FAMS"
1970 G$=INKEY$:IF G$="" THEN 19
70
1980 A=INSTR(1,F$,G$)
1990 IF A<=0 THEN 1960
2000 IF I<>6 AND I<>5 AND A=1 T
HEN GOSUB 2540 DRAW"BM64,152":A
$="NAO HA ARCOS":GOSUB 3190:GOS
UB 2410:GOTO 1910
2010 IF A=4 THEN GOSUB 2440:RET
URN
2020 T(I,1)=A
2030 IF A=3 THEN GOSUB 2050
2040 RETURN
```

As linhas 1920 a 1950 mostram as opções na tela. As linhas 1965 a 1985 obtêm a resposta do teclado e colocam o número equivalente à ordem selecionada na variável **A**: 0 é "fogo", 1 é "alto", 2 é "marche" e 3 é "status".

A ordem de atirar só será possível por meio de uma rotina do próximo artigo da série. As opções de marchar e de status serão dadas pelas próximas rotinas. A ordem "ALTO" não precisa de uma rotina especial, já que nada acontece com a unidade. O elemento da matriz da tropa que indica a direção em que se movimentava a unidade não deve ser modificado porque é utilizado na rotina de combate.

A linha 2020 coloca o valor da variável **A** na matriz da tropa; o número da ordem corresponde ao primeiro elemento da matriz.

UMA NOVA DIREÇÃO

Se o jogador ordenar a uma unidade que se movimente, o programa solicitará uma direção. Esta rotina cuida das opções possíveis:

```
2050 REM  Direcao
2055 GOSUB 2540
2060 PRINT AT 17,0;"Para onde (
NSLO) ?"
2065 LET g=0
2070 REM loop
2080 LET g$=INKEY$: IF g$="" TH
EN  GOTO 2080
2090 IF CODE (g$)>90 THEN  LET
g$=CHR$ (CODE (g$)-32)
2095 FOR k=1 TO 4
2100 IF i$(k)=g$ THEN  LET g=k
2105 NEXT k
2110 IF g=0 THEN  GOTO 2070
2120 LET T(i,2)=g
2130 RETURN
```



```
2050 REM DIREÇAO
2055 GOSUB 2540
2060 LOCATE 0,18:PRINT "PARA ON
DE (NSLO)?"
2065 G=0
2070 REM
2075 REM
2080 G$=INKEY$:IF G$="" THEN 20
80
2100 G=INSTR(1,I$,G$)
2110 IF G=0 THEN 2070
2120 T(I,2)=G
2130 RETURN
```



```
2050  REM  DIRECAO
2055  GOSUB 2540
2060  VTAB 21: PRINT "PARA ONDE
 (NSLO)?"
2065  GOSUB 30000
2070 G = 0
2080  GET G$
2090  IF  ASC (G$) > 90 THEN G$
 =  CHR$ ( ASC (G$) - 32)
2095  FOR K = 1 TO 4
2100  IF  MID$ (I$,K,1) = G$ TH
EN G = K
2105  NEXT K
2110  IF G = 0 THEN 2070
2120 T(I,2) = G
2130  RETURN
```



Modificações a fazer no programa:

```
2065  REM
```



```
2050 REM DIRECAO
2055 GOSUB 2540
2060 DRAW"BM0,144":A$="QUAL A D
IRECAO        N S L O ":GOSUB 31
90
2065 G=0
2070 REM
2080 G$=INKEY$:IF G$="" THEN 20
80
2100 G=INSTR(1,I$,G$)
2110 IF G<=0 THEN 2070
2120 T(I,2)=G
2130 RETURN
```

A rotina verifica se a letra teclada pelo jogador é N, S, L ou O. Se for uma delas, a linha 2120 colocará o número do movimento equivalente no segundo elemento da matriz da tropa.

STATUS

Para conferir a situação de uma unidade, o jogador terá esta rotina.



```
2440 REM status
2450 GOSUB 2540
2470 PRINT AT 18,0;"Arma: ";w$(
T(i,3))
2480 PRINT AT 18,15;"Armadura:
";a$(T(i,4))
2490 PRINT AT 19,0;"Poder: ";T(
i,7)
2500 PRINT AT 19,14;"Moral: ";m
$(T(i,5))
2510 PRINT AT 20,0;"Terreno: ";
r$(m(T(i,8),T(i,9))+1)
2520 GOSUB 2410
2530 RETURN
2560 PRINT AT 17,0;"UNIDADE ";i
;"     Tipo: ";t$(i)
```



```
2440 REM STATUS
```



```
2450 GOSUB 2540
2460 LOCATE 0,18:PRINT "UNIDADE
";I,"TIPO:";T$(I)
2470 LOCATE 0,19:PRINT "ARMA:";
W$(T(I,3))
2480 LOCATE 15,19:PRINT "ARMADU
RA:";A$(T(I,4))
2490 LOCATE 0,20:PRINT "PODER:"
;T(I,7)
2500 LOCATE 15,20:PRINT "MORAL:
";M$(T(I,5))
2510 LOCATE 0,20:PRINT "POSIÇÃO
:";R$(M(T(I,8),T(I,9))+1)
2520 GOSUB 2410
2530 RETURN
```



```
2440  REM  STATUS
2450  GOSUB 2540
2460  VTAB 21: PRINT "UNIDADE "
;I,".";"TIPO: "T$(I)
2470  PRINT "ARMA: ";W$(T(I,3))
;".";
2480  PRINT "ARMADURA: ";A$(T(I
,4));"."
2490  PRINT "PODER: ";T(I,7);".
";
2500  PRINT "MORAL: ";M$(T(I,5)
);"."
2510  PRINT "TERRENO: ";R$(M(T(
I,8),T(I,9)) + 1)
2520  GOSUB 2410
2530  RETURN
```



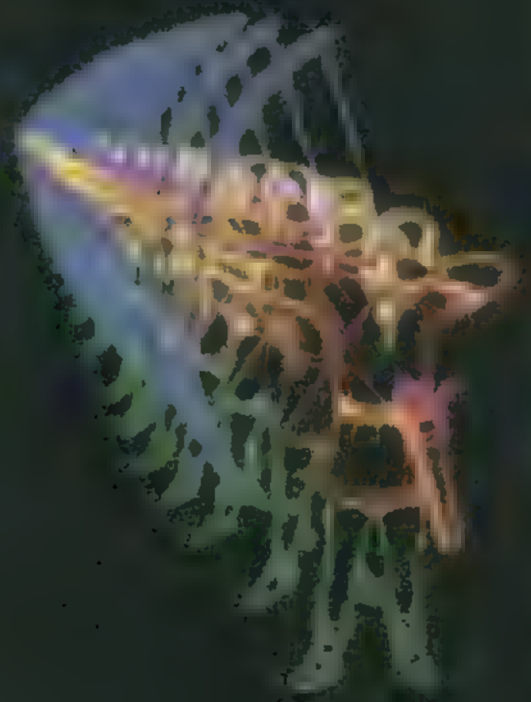
Modificações para o TK-2000:

```
2545  HCOLOR= 0: FOR ER = 160 T
O 191
2546  HPLOT 0,ER TO 279,ER: NEX
T
```



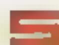
```
2440 REM  STATUS
2450 GOSUB 2540
2460 DRAW"BM0,144":A$="UNIDADE"
+STR$(I)+"    TIPO "+T$(I):GOSU
B 3190
2470 DRAW"BM0,152":A$="ARMA "+W
$(T(I,3)):GOSUB 3190
2480 DRAW"BM120,152":A$="ARMADU
RA "+A$(T(I,4)):GOSUB 3190
2490 DRAW"BM0,160":A$="PODER "+
SR$(T(I,7)):GOSUB 3190
2500 DRAW"BM120,160":A$="MORAL
"+M$(T(I,5)):GOSUB 3190
2510 DRAW"BM0,168":A$="TERRENO
"+R$(M(T(I,8),T(I,9))+1):GOSUB
3190
2520 GOSUB 2410
2530 RETURN
```

Todos os elementos pertencentes à matriz da tropa (ou as palavras correspondentes) que se referem à unidade que o jogador está querendo conferir são mostrados na tela.



O EFEITO DAS ORDENS

Esta rotina informa ao jogador quando cada unidade obedece às ordens.



```
1020 REM  cumpre
1030 FOR i=1 TO 16
1032 IF t(i,1)>3 THEN  GOTO 114
0
1035 INK 0: GOSUB 2540: PRINT A
T 17,0;"Unidade ";i;" entra em
acao !"
1040 LET cl=1: IF i>8 THEN  LET
 cl=2: INK cl
1050 IF T(i,1)=3 THEN  LET b=1:
 GOSUB 1160
1055 IF T(i,1)=2 THEN  GOTO 114
0
1060 IF T(i,1)=1 THEN  LET sh=i
: GOSUB 1710: GOTO 1140
1070 FOR f=-1 TO 1
1080 FOR g=-1 TO 1
1090 FOR e=1 TO 16
1100 IF (T(i,8)+f=T(e,8)) AND (
T(i,9)+g=T(e,9)) AND T(e,1)<>5
THEN  LET us=i: LET th=e: GOSUB
 1510
1110 NEXT e
1120 NEXT g
1130 NEXT f
1140 NEXT i
1150 RETURN
```



```
1020 REM CUMPRE
1030 FOR I=1 TO 16
1032 IF T(I,1)>3 THEN 1140
1035 GOSUB 2540:LOCATE 0,18:PRI
NT "UNIDADE";I;"OBEDECE"
1040 IF T(I,1)>3 THEN 1140
1050 IF T(I,1)=3 THEN B=I:GOSUB
 1160
1055 IF T(I,1)=2 THEN 1140
1060 IF T(I,1)=1 THEN SH=I:GOSU
B 1710:GOTO 1140
1070 FOR F=-1 TO 1
1080 FOR G=-1 TO 1
1090 FOR E=1 TO 16
1100 IF (T(I,8)+F=T(E,8)) AND(T
(I,9)+G=T(E,9)) AND T(E,1)<>5 T
HEN US=I:TH=E:GOSUB 1510
1110 NEXT E
1120 NEXT G
1130 NEXT F
1140 NEXT I
1150 RETURN
```



```
1020  REM  CUMPRE
1030  FOR I = 1 TO 16
1032  IF T(I,1) > 3 THEN 1140
1035  GOSUB 2540: VTAB 21: PRIN
T "A UNIDADE ";I;" OBEDECE": GO
SUB 30000
1040  LET CL = 0: IF I > 8 THEN
 CL = 5
1050  IF T(I,1) = 3 THEN B = I:
 GOSUB 1160
1055  IF T(I,1) = 2 THEN 1140
1060  IF T(I,1) = 1 THEN SH = I
 : GOSUB 1710: GOTO 1140
1070  FOR F =  - 1 TO 1
1080  FOR G =  - 1 TO 1
1090  FOR E = 1 TO 16
1100  IF (T(I,8) + F = T(E,8))
AND (T(I,9) + G = T(E,9)) AND T
(E,1) <  > 5 THEN US = I:TH = E
 : GOSUB 1510
1110  NEXT E
1120  NEXT G
1130  NEXT F
1140  NEXT I
1150  RETURN
```



Modificações necessárias:

```
1035  GOSUB 2540: VTAB 21: PRIN
T "A UNIDADE ";I;" OBEDECE"
```



```
1020 REM CUMPRE
1030 FOR I=1 TO 16
1032 IF T(I,1)>3 THEN 1140
1035 COLOR 4:GOSUB 2540:DRAW "B
M0,144":A$="UNIDADE"+STR$(I)+"E
NTRA EM ACAO !":GOSUB 3190
1040 CL=3:IF I>8 THEN CL=4:COLO
R CL
1050 IF T(I,1)=3 THEN B=1:GOSUB
 1160
1055 IF T(I,1)=2 THEN 1140
1060 IF T(I,1)=1 THEN SH=I:GOSU
B 1710:GOTO 1140
1070 FOR F=-1 TO 1
1080 FOR G= 1 TO 1
1090 FOR E=1 TO 16
1100 IF (T(I,8)+F=T(E,8)) AND (
T(I,9)+G=T(E,9)) AND T(E,1)<>5
THEN US=1:TH=E:GOSUB 1510
1110 NEXT E,G,F
1140 NEXT I
1150 RETURN
```

O funcionamento dessa rotina é particularmente simples. Ela toma cada uma das unidades e executa as ordens. Só haverá combate se uma das unidades se mover (a que se moveu por último é considerada atacante). Quando uma ordem de "alto" é detectada, tudo permanece como está, passando-se à próxima opção. Quando ordens de "fogo" são encontradas, a rotina de tiro é chamada. Se as ordens forem de "marche", será chamada a rotina **MOVE**.

Após a realização de um movimento, a rotina verifica as posições adjacentes à nova localização da unidade em busca de tropas inimigas (linhas 1070 a 1150). Se alguma delas for encontrada, será chamada a rotina de combate, que apresentaremos no próximo artigo.

A VEZ DO COMPUTADOR

Por enquanto, o computador dará ordens ao acaso. Na última parte desta série, veremos como transformá-lo em um adversário inteligente. A rotina seguinte limita-se a tornar o programa possível de ser jogado.



```
2140 REM inimigo
2150 LET T(e,2)=3
2160 LET T(e,1)=FN r(3)
2170 IF T(e,1)=1 AND T(e,3)<>2
THEN  GOTO 2160
2180 IF T(e,1)=3 THEN  IF FN r(
2)=1 THEN  LET T(e,2)=FN r(4)
2190 RETURN
```



```
2140 REM INIMIGO
2150 T(E,2)=3
2160 T(E,1)=FN R(3)
2170 IF T(E,1)=1 AND T(E,3)<>2
THEN 2160
2180 IF T(E,2)=3 THEN IF FN R(2
)=1 THEN T(E,2)=FN R(4)
2190 RETURN
```



```
2140  REM  INIMIGO
2150  T(E,2) = 3
2160  T(E,1) =  FN R(3)
2170  IF T(E,1) = 1 AND T(E,3)
<  > 2 THEN 2160
2180  IF T(E,1) = 3 THEN  IF  F
N R(2) = 1 THEN T(E,2) =  FN R(
4)
2190  RETURN
```



```
2140 REM  INIMIGO
2150 T(E,2)=3
2160 T(E,1)=RND(3)
2170 IF T(E,1)=1 AND T(E,3)<>2
THEN 2160
2180 IF T(E,1)=3 AND RND(2)=1 T
HEN T(E,2)=RND(4)
2190 RETURN
```

A rotina gera um número aleatório na linha 2160 que decide o tipo de ação das unidades do computador na jogada. Se o computador decidir mover alguma unidade, haverá uma tendência ao movimento em direção sul (linha 2150).

UM AMPLIADOR GRÁFICO

- COMO DESENHAR
- AJUSTE A ESCALA
- COMO AMPLIAR E REDUZIR
- DETALHES ESCONDIDOS
- DESENHOS DE PRECISÃO

O programa deste artigo pode reduzir um desenho a proporções diminutas ou expandi-lo em escala astronômica. Utilize-o como jogo ou para realizar desenhos muito detalhados.

O programa que acompanha este artigo é muito mais divertido do que os programas aplicativos habituais. De fato, ele pode ser usado, seja como jogo, seja como aplicação.

Ele permite desenhar e, em seguida, ampliar ou reduzir, até um nível microscópico, partes de uma figura para obter maiores detalhes. É possível, além disso, mudar a escala a qualquer momento, usando ampliações de milhares de vezes ou até mesmo de centenas de milhares. Assim, uma enorme quantidade de detalhes que seriam imperceptíveis a olho nu numa figura de proporções reduzidas aparece de modo bem claro quando ela é submetida a uma ampliação por meio de um *zoom*.

Imagine-se com um poderoso microscópio, penetrando cada vez mais em direção à estrutura da imagem. Os detalhes desta aparecerão à medida que a ampliação for aumentando e, inversamente, desaparecerão quando ela sofrer uma redução de certa magnitude.

Para tornar ainda mais claro esse processo, podemos considerar o seguinte exemplo: se começarmos a fotografar nossa casa e nos afastarmos pouco a pouco até uma distância de centenas de milhares de quilômetros, os objetos — primeiro a casa, depois a rua, o bairro, a cidade, o país e mesmo a Terra — desaparecerão gradualmente das imagens retidas pela câmara. Tais efeitos gráficos podem ser conseguidos com o programa deste artigo.

Uma maneira de usar o programa é transformá-lo em um jogo para duas pessoas. Uma delas faz um desenho que esconde, em algum lugar discreto, um "tesouro", de tamanho diminuto. A outra deve procurar o tesouro. Essa tarefa pode se revelar surpreendentemente difícil. Imaginemos que o jogador comece sua busca em um quadrado de 10 cm de lado. Se este for ampliado 5 000 vezes, o quadrado original se transformará em uma área enorme de cerca de 250 000 m^2 — espaço suficiente para esconder qualquer coisa.

O programa se presta ainda a aplicações em diferentes áreas profissionais. Desenhos minuciosos, por exemplo, podem ser produzidos traçando-se os detalhes em uma escala de grandes proporções e depois reduzindo-se a figura para um tamanho normal. Desenhos técnicos muito precisos podem ser obtidos trabalhando-se em um setor de cada vez. O programa também é útil como elemento de apoio no campo pedagógico. Tomemos como exemplo uma lição geográfica. A posição das maiores cidades pode ser marcada no mapa em uma escala muito reduzida. No tamanho normal, ficariam só como pontos residuais e apareceriam em detalhe apenas quando se usasse um *zoom* no local adequado.

O programa oferece também a possibilidade de se escrever nomes. Embora não permita o uso de letras normais, é possível desenhá-las. Para isso, recorra a uma escala bem grande e depois reduza as letras para o tamanho adequado.





Ampliando este desenho...



você poderá olhar através da vidraça...



e verificar que há um livro sobre a mesa!

COMO EXECUTAR O PROGRAMA

Ao ser executado, o programa apresentará a tela em branco e um pequeno cursor no centro dela. Você encontrará também dois números: o da esquerda indica a escala da tela atual — 1 para começar; o da direita mostra o número de linhas desenhadas. Esse número é limitado a um máximo de seiscentos no início do programa, que é controlado pelo teclado e muito fácil de usar.

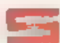
No MSX, no Spectrum e no TRS-Color, o cursor é controlado pelas setas do teclado. Nos outros micros, as teclas são: I, para cima; M, para baixo; J, para a esquerda e K, para a direita. Pressionando-se simultaneamente essas teclas — **<SYMBOL-SHIFT>** no Spectrum, ou **<CLEAR>** no TRS-Color — acelera-se o cursor.

As outras teclas usadas são L, M, C, D, E, S e Z. Veja o que elas fazem: L desenha uma linha junto ao cursor; M move o cursor sem desenhar (nos micros da linha Apple, use ; em vez de M para mover o cursor). Pressionar C leva o cursor ao centro da tela; D cancela linhas. O número de linhas a ser apagado será perguntado pelo computador. Pressione S para ter acesso a um arquivo de saída que permita a armazenagem de seu desenho em disco ou fita.

Agora, a parte mais interessante. A tecla Z (para zoom) permite a alteração da escala. É digitada sempre com um número: 2, por exemplo, dobra a escala e 0.5 a reduz ao meio. O desenho é refeito na nova escala e *centrado no cursor*. Assim, você deve deixar o cursor sempre no meio da área que deseja ampliar.

Usar 0 como escala não altera o desenho, enquanto 1 redesenha a figura centrada no cursor, reproduzindo-a na escala anterior (no Apple, 0 centra o desenho e 1 retorna a imagem para a escala 1). Um número negativo produz uma imagem em espelho do desenho. As mudanças de escala são cumulativas. Assim, um aumento de 2 seguido de outro de 2 leva a ampliação para 4.

Note que uma escala muito pequena no TRS-Color, como 0.0001, é convertida para 10E-4, mas aparece na tela como 104. Esta é uma limitação da rotina que desenha números na tela e não afeta a maneira com que o desenho é feito.



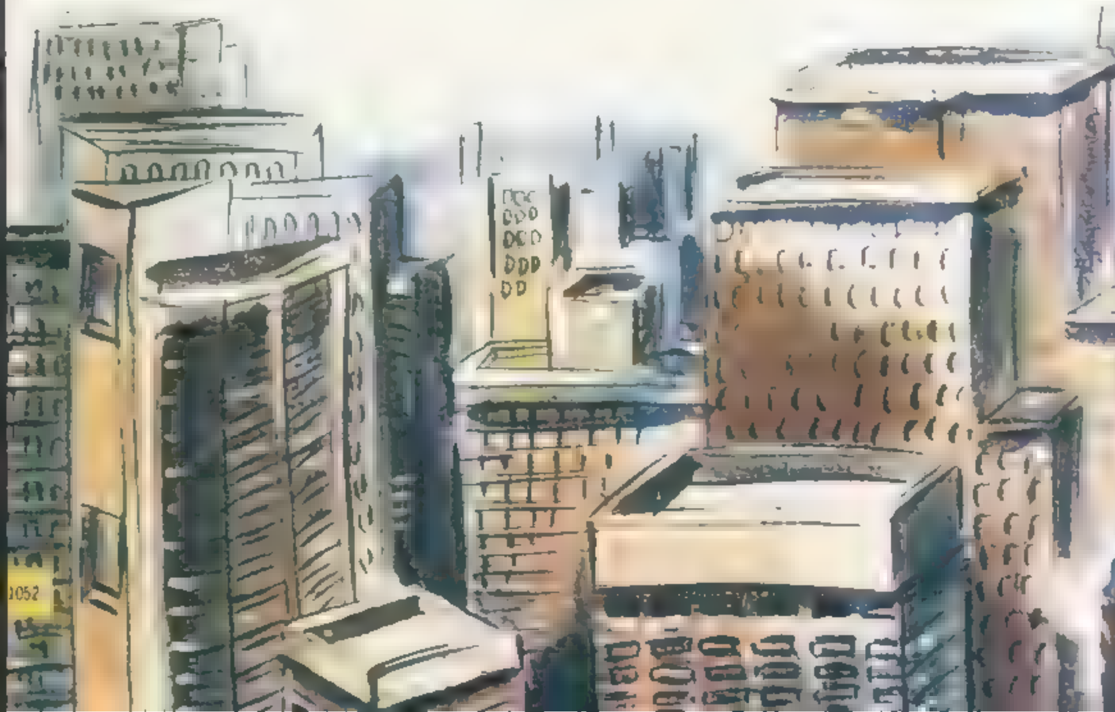
```
 10 DEF FN a(x)=(x/256)+128
 20 DEF FN b(x)=(x/256)+85
 30 BORDER 0· INK 7: PAPER 0:
CLS
 50 LET ln=1: LET sc=1: LET l=
600
 60 LET f=0: LET x=0: LET y=0:
LET zoom=1
 70 DIM a(l+1): DIM b(l): DIM
c(l): DIM d(l)
 80 LET a(l)=x· LET b(l)=y
 90 LET c(l)=x: LET d(l)=y
100 LET i$=INKEY$
110 IF i$="l" AND f AND l>ln
THEN  SOUND 0.1,0: PLOT FN a(a
(ln)),FN b(b(ln)): DRAW FN a(c
(ln))-PEEK 23677,FN b(d(ln))-
PEEK 23678: LET ln=ln+1: LET a
(ln)=x: LET b(ln)=y: LET c(ln)
=x: LET d(ln)=y: LET f=0
120 IF i$="c" THEN  LET x=0:
LET y=0: LET c(ln)=x: LET d(ln
)=y
130 IF i$="m" THEN  LET a(ln)=
x: LET b(ln)=y
140 IF i$="i" THEN  GOSUB 420
150 IF i$="o" THEN  GOSUB 600
160 IF i$="d" THEN  GOSUB 710
170 IF i$="z" THEN  LET lv=0:
GOSUB 840
180 LET sp=256: IF CODE INKEY$
<=41 THEN  LET sp=2048
190 IF INKEY$="8" OR INKEY$="(
" AND x>-32768+sp THEN  LET f=
1: LET x=x+sp: LET c(ln)=c(ln)
+sp
200 IF INKEY$="5" OR INKEY$="%
" AND x<32678-sp THEN  LET f=
1: LET x=x-sp: LET c(ln)=c(ln)
-sp
210 IF INKEY$="7" OR INKEY$="'
" AND y>-22400+sp THEN  LET f=
1: LET y=y+sp· LET d(ln)=d(ln
)+sp
220 IF INKEY$="6" OR INKEY$="&
" AND y<22400-sp THEN  LET f=1
: LET y=y-sp  LET d(ln)=d(ln)-
sp
230 GOSUB 250
240 GOTO 100
250 PRINT AT 1.2: INVERSE 1;"
ESCALA:";: LET a$=STR$ sc· IF
LEN a$>4 THEN  LET a$=a$( TO 4
)
255 PRINT  INVERSE 1;a$;AT 1,
18;" LINHAS·";ln-1
280 LET bx=FN a(a(ln)): LET by
=FN b(b(ln)): LET ex=FN a(c(ln
)): LET ey=FN b(d(ln))
290 PLOT bx,by: PLOT ex,ey
310 PLOT  OVER 1;bx,by  PLOT
OVER 1;ex,ey
320 RETURN
420 CLS
430 INPUT "NOME DO ARQUIVO ?";
f$
435 IF f$="" THEN  GOTO 430
440 LET ln=1: LET zoom=1· LET
sc=1
460 LOAD f$ DATA a()
470 LOAD f$ DATA b()
480 LOAD f$ DATA c()
485 LOAD f$ DATA d()
490 LET ln=a(601)
590 RETURN
600 CLS
610 INPUT "NOME DO ARQUIVO ?",
f$
615 IF f$="" THEN  GOTO 610
620 LET lv 1· GOSUB 840
625 LET a(601)=ln
630 SAVE f$ DATA a()
640 SAVE f$ DATA b()
650 SAVE f$ DATA c()
660 SAVE f$ DATA d()
700 RETURN
710 INPUT "NUMERO DE LINHAS A
APAGAR ?"'k
750 IF k=0 OR ln-k<=0 THEN
GOTO 830
755 LET ln=ln-k
760 LET x a(ln)· LET y=b(ln)
780 CLS
790 LET c(ln)=x: LET d(ln)=y
800 IF ln=1 THEN  GOTO 830
```

```
810 IF ABS (c(ln-1))<32767 AND
ABS (d(ln-1))<22399 THEN  LET
x=0: LET y=0
820 LET lv=2: GOSUB 840
830 RETURN
840 IF ln=0 THEN  RETURN
850 IF lv=1 THEN  LET zoom=1/
sc  CLS : GOTO 920
860 IF lv=2 THEN  LET zoom=1:
GOTO 920
870 SOUND .1,10: INPUT "DIGITE
A ESCALA  -  ZOOM ";zoom
890 IF zoom=0 THEN  GOTO 1020
910 CLS
920 FOR u=1 TO ln-1
930 LET a(u)=(a(u)-x)*zoom:
LET b(u)=(b(u)-y)*zoom
940 LET c(u)=(c(u)-x)*zoom:
LET d(u)=(d(u)-y)*zoom
950 IF ABS (a(u))<32768 AND
ABS (b(u))<22400 AND ABS (c(u)
)<32768 AND ABS (d(u))<22400
THEN  PLOT FN a(a(u)),FN b(b(u
)): DRAW FN a(c(u))-PEEK 23677
,FN b(d(u))-PEEK 23678
960 NEXT u
970 LET a(u)=(a(u)-x)*zoom:
LET b(u)=(b(u)-y)*zoom
980 LET c(u)=(c(u)-x)*zoom:
LET d(u)=(d(u)-y)*zoom
990 LET x=c(ln): LET y=d(ln)
1000 IF ABS (a(ln))>32767 OR AB
S (b(ln))>22399 THEN  LET a(ln)
=x: LET b(ln)=y
1010 LET sc=sc*zoom
1030 RETURN
```



```
10 PMODE 4,1:COLOR 0,1:PCLS:SCR
EEN 1,0:V=247
20 DIM NU$(10):FOR I=0 TO 10:RE
AD NU$(I):NEXT
30 DEF FN A(X)=(X/256)+128
40 DEF FN B(X)=(X/256)+96
50 LN=0:SC=1:L=600
60 X=0:Y=0:ZOOM=1
70 DIM BX(L),BY(L),EX(L),EY(L)
80 BX(0)=X:BY(0)=Y
90 EX(0)=X:EY(0)=Y
100 I$=INKEY$
110 IF I$="L" AND F AND L>LN TH
EN LINE(FNA(BX(LN)),FNB(BY(LN))
)-(FNA(EX(LN)),FNB(EY(LN))),PSE
T:LN=LN+1:BX(LN)=X:BY(LN)=Y:EX(
LN)=X:EY(LN)=Y:F=0
120 IF I$="C" THEN X=0:Y=0:EX(L
N)=X:EY(LN)=Y
```

```
130 IF I$="M" THEN BX(LN)=X BY(
LN)=Y
140 IF I$="I" GOSUB 420
150 IF I$="O" GOSUB 600
160 IF I$="D" GOSUB 710
170 IF I$="Z" THEN LV=0:GOSUB 8
40
180 IF PEEK(339)=191 THEN SP-20
48 ELSE SP=256
190 IF PEEK(343)=V AND X>-32768
+SP THEN F= 1 X=X SP EX(LN)=EX(
LN)-SP
200 IF PEEK(344)=V AND X<32768-
SP THEN F= 1 X X+SP EX(LN)=EX(L
N)+SP
210 IF PEEK(341)=V AND Y> 24576
+SP THEN F= 1 Y=Y-SP.EY(LN)=EY(
LN)-SP
220 IF PEEK(342)=V AND Y<24576
SP THEN F=-1 Y=Y+SP EY(LN)=EYT(
LN)+SP
230 GOSUB 250
240 GOTO 100
250 SS$=STR$(SC)
260 DRAW"BM0,183"·GOSUB 330
270 SS$=STR$(LN) DRAW"BM200,183
" GOSUB 330
280 BX=FNA(BX(LN))·BY=FNB(BY(LN
)) EX=FNA(EX(LN)).EY=FNB(EY(LN)
)
290 PSET(BX,BY)·PSET (EX,EY)
300 FOR D=1 TO 50·NEXT D
310 PRESET(BX,BY) PRESET(EX.EY)
320 RETURN
330 FOR I=1 TO LEN(SS$)
340 DI=ASC(MID$(SS$,I,1)) 48
350 IF DI=-2 THEN DI=10
360 IF DI<0 OR DI>10 THEN 380
370 DRAW"C1,XNU$(8);C0;BL8"+NU$
(DI)+"BR2"
380 NEXT I
390 RETURN
400 DATA R6D8L6U8BR8,BR6ND8BR2,
R6D4L6D4R6BR2BU8,R6D4NL3D4NL6BR
2BU8,D4R6D4U8BR2,NR6D4R6D4L6BE8
410 DATA D8R6U4L6U4BR8,R6ND8BR2
,R6D8L6U8D4R6U4BR2.D4R6D4U8L6BR
8,BR3BD8NR1BR5BU8
420 CLS
430 PRINT @256,"";:LINE INPUT"N
OME DO ARQUIVO A CARREGAR  ?";F
$
440 LN=0·ZOOM=1:SC=1
450 PCLS·SCREEN 1,0
460 OPEN"I",#-1,F$
470 INPUT#-1.N$
480 LN=VAL(N$)
490 FOR U=0 TO LN-1
500 INPUT #-1,BX$,BY$.EX$.EY$
510 BX(U)=VAL(BX$) BY(U)=VAL(BY
$) EX(U) VAL(EX$) EY(U)=VAL(EY$
)
520 IF ABS(BX(U))<32768 AND ABS
(BY(U))<24576 AND ABS(EX(U))<32
768 AND ABS(EY(U))<24576 THEN L
INE (FNA(BX(U)),FNB(BY(U))) (FN
A(EX(U)).FNB(EY(U))),PSET
530 NEXT
540 INPUT #-1,BX$.BY$.EX$,EY$
550 BX(U)=VAL(BX$)·BY(U) VAL(BY
$).EX(U)=VAL(EX$):EY(U)=VAL(EY$
)
560 X=EX(LN):Y=EY(LN)
570 CLOSE #-1
580 SCREEN 1.0
590 RETURN
600 CLS
610 PRINT @256,"";:LINE INPUT "
NOME DO ARQUIVO A SALVAR  ?";F$
620 LV=1:GOSUB 840
630 OPEN "O",#-1,F$
640 PRINT#-1,STR$(LN)
650 FOR U=0 TO LN
660 PRINT #-1,STR$(BX(U)).STR$(
BY(U)),STR$(EX(U)).STR$(EY(U))
679 NEXT
```



```
680 CLOSE #-1
690 SCREEN 1,0
700 RETURN
710 CLS
720 PRINT @256,"QUANTAS LINHAS
QUER APAGAR ";
730 INPUT K
740 SCREEN 1,0
750 IF K=0 OR LN-K<0 THEN 830
760 LN=LN-K
770 X=BX(LN):Y=BY(LN)
780 PCLS
790 EX(LN)=X:EY(LN)=Y
800 IF LN=0 THEN 830
810 IF ABS(EX(LN-1))<32767 AND
ABS(EY(LN-1))<24576 THEN X=0:Y=
0
820 LV=2:GOSUB 840
830 SCREEN 1,0 RETURN
840 IF LN=0 THEN RETURN
850 IF LV=1 THEN ZOOM=1/SC.PCLS
·GOTO 920
860 IF LV=2 THEN ZOOM=1.GOTO 92
0
870 CLS
880 PRINT @256," DIGITE A ESCAL
A / ZOOM ";
890 INPUT ZOOM
900 IF ZOOM=0 THEN 1020
910 PCLS:SCREEN 1,0
920 FOR U=0 TO LN-1
930 BX(U)=(BX(U)-X)*ZOOM·BY(U)=
(BY(U)-Y)*ZOOM
940 EX(U)=(EX(U)-X)*ZOOM·EY(U)=
(EY(U)-Y)*ZOOM
950 IF ABS(BX(U))<32768 AND ABS
 (BY(U))<24576 AND ABS(EX(U))<3
2768 AND ABS(EY(U))<24576 THEN
LINE(FNA(BX(U)),FNB(BY(U)))-(FN
A(EX(U)),FNB(EY(U))),PSET
960 NEXT
970 BX(U)=(BX(U)-X)*ZOMM·BY(U)=
(BY(U)-Y)*ZOOM
980 EX(U)=(EX(U) X)*ZOOM:EY(U)=
(EY(U)-Y)*ZOOM
990 X=EX(LN).Y=EY(LN)
1000 IF ABS(BX(LN))>32767 OR AB
S(BY(LN))>24576 THEN BX(LN)=X:B
Y(LN)=Y
1010 SC=SC*ZOOM
1020 SCREEN 1.0
1030 RETURN
```



```
5  LOMEM: 16384
10  HGR   HCOLOR= 3· HOME :D$ =
  CHR$ (13) +  CHR$ (4)
30  DEF  FN A(X) = (X / 256) +
128
40  DEF  FN B(X) = (X / 256) +
96
50 LN = 0:SC = 1:L = 600:LM = 8
192
60 X = 0:Y = 0·ZOOM - 1·SP = 25
6
70  DIM BX(L),BY(L),EX(L),EY(L)

80 BX(0) = X·BY(0) = Y
90 EX(0) = X·EY(0) = Y
95  GOSUB 240
100  GET I$
110  IF I$ = "L" AND F AND L >
LN THEN  HPLOT  FN A(BX(LN)), F
N B(BY(LN)) TO  FN A(EX(LN)), F
N B(EY(LN))· VTAB 21:LN = LN +
1:BX(LN)   X·BY(LN) = Y·EX(LN)
= X:EY(LN)   Y·F   0:M1 = 0·M2
= 0: GOSUB 320
120  IF I$ = "C" THEN X = 0:Y =
 0:EX(LN) = X·EY(LN)   Y
130  IF I$ = ";" THEN BX(LN) =
X·BY(LN) = Y
140  IF I$ = "E" THEN  GOSUB 42
0
```

```
150  IF I$ = "S" THEN  GOSUB 60
0
160  IF I$ = "D" THEN  GOSUB 71
0
170  IF I$ = "Z" THEN LV = 0: G
OSUB 840
180  IF I$ = "J" AND X >  - 327
68 THEN F =  - 1:X  X - SP:EX(
LN) = EX(LN) - SP
190  IF I$ = "K" AND X < 32768
- SP THEN F =  - 1:X = X + SP:E
X(LN) = EX(LN) + SP
200  IF I$ = "I" AND Y >  - 245
76 + SP THEN F =  - 1:Y = Y - S
P:EY(LN) = EY(LN) - SP
210  IF I$ = "M" AND Y < 16128
THEN F =  - 1:Y = Y + SP:EY(LN)
 = EY(LN) + SP
220  GOSUB 240
230  GOTO 100
240  IF M <  > 0 THEN  POKE M,M
E: POKE M0,MA
245  VTAB 21: HTAB 1: CALL  - 9
58: PRINT "ESCALA:";SC,"LINHAS:
";LN
250 BX =  FN A(BX(LN)):BY =  FN
 B(BY(LN)):EX =  FN A(EX(LN)):E
Y =  FN B(EY(LN))
260 LX =  INT (EY / 64):CX =  I
NT (EY / 8) - (8 *  INT (EY / 6
4)):TX = EY - (8 *  INT (EY / 8
))
270 M = LM + (LX * 40) + (CX *
128) + (TX * 1024)
280 M = M + ( INT (EX / 7))
290 LX =  INT (BY / 64):CX =  I
NT (BY / 8) - (8 *  INT (BY / 6
4)):TX = BY - (8 *  INT (BY / 8
))
300 M0 = LM + (LX * 40) + (CX *
 128) + (TX * 1024)
310 M0 = M0 + ( INT (BX / 7))
320  IF M <  > M1 THEN M1 = M:M
E =  PEEK (M)
330  IF M0 <  > M2 THEN M2 = M0
:MA =  PEEK (M0)
340  HPLOT EX,EY  HPLOT BX,BY
350  RETURN
420  TEXT : HOME
430  INPUT "NOME DO ARQUIVO A S
ER LIDO:";F$
440 LN = 0:ZOOM = 1:SC = 1
450  HGR
460  PRINT D$;"OPEN ";F$
470  PRINT D$;"READ ";F$
480  INPUT LN
490  FOR U = 0 TO LN - 1
500  INPUT BX(U),BY(U),EX(U),EY
(U)
520  IF  ABS (BX(U)) < 32768 AN
D  ABS (BY(U)) < 24576 AND  ABS
 (EX(U)) < 32768 AND  ABS (EY(U
)) < 24576 THEN  HPLOT  FN A(BX
(U)), FN B(BY(U)) TO  FN A(EX(U
)), FN B(EY(U))
530  NEXT
540  INPUT BX(U),BY(U),EX(U),EY
(U)
560 X = EX(LN):Y = EY(LN)
570  PRINT D$;"CLOSE"
590  RETURN
600  TEXT : HOME
610  INPUT "NOME DO ARQUIVO A S
ER GRAVADO:";F$
620 LV = 1: GOSUB 840
630  PRINT D$;"OPEN ";F$
635  PRINT D$;"WRITE ";F$
640  PRINT LN
650  FOR U = 0 TO LN
660  PRINT BX(U): PRINT BY(U):
PRINT EX(U): PRINT EY(U)
670  NEXT
680  PRINT D$;"CLOSE"
700  RETURN
710  TEXT : HOME
720  INPUT "QUANTAS LINHAS QUER
 APAGAR? ";K
750  IF K = 0 OR LN - K < 0 THE
N 830
760 LN = LN - K
770 X = BX(LN):Y = BY(LN)
780  HGR
790 EX(LN) = X:EY(LN) = Y
800  IF LN = 0 THEN 830
810  IF  ABS (EX(LN - 1)) < 327
67 AND  ABS (EY(LN - 1)) < 2457
6 THEN X = 0:Y = 0
820 LV = 2: GOSUB 840
830  RETURN
840  IF LN = 0 THEN  RETURN
850  IF LV = 1 THEN ZOOM = 1 /
SC: GOTO 910
860  IF LV = 2 THEN ZOOM = 1: G
OTO 910
870  VTAB 23
880  INPUT "ESCALA DO ZOOM ? ";
ZOOM
900  IF ZOOM = 1 THEN LV = 1: G
OTO 850
905  IF ZOOM = 0 THEN ZOOM = 1
910  HGR
920  FOR U = 0 TO LN - 1
930 BX(U) = (BX(U) - X) * ZOOM:
BY(U) = (BY(U) - Y) * ZOOM
940 EX(U) = (EX(U) - X) * ZOOM:
EY(U) = (EY(U) - Y) * ZOOM
950  IF  ABS (BX(U)) < 32768 AN
D  ABS (BY(U)) < 24576 AND  ABS
 (EX(U)) < 32768 AND  ABS (EY(U
)) < 24576 THEN  HPLOT  FN A(BX
(U)), FN B(BY(U)) TO  FN A(EX(U
)), FN B(EY(U))
960  NEXT
970 BX(U) = (BX(U) - X) * ZOOM:
BY(U) = (BY(U) - Y) * ZOOM
980 EX(U) = (EX(U) - X) * ZOOM:
EY(U) = (EY(U) - Y) * ZOOM
990 X = EX(LN):Y = EY(LN)
1000  IF  ABS (BX(LN)) > 32767
OR  ABS (BY(LN)) > 24576 THEN B
X(LN) = X:BY(LN) = Y
1010 SC = SC * ZOOM
1030  RETURN
```

MSX

```
5 MAXFILES=2:CLEAR 2000
10 OPEN "GRP:" FOR OUTPUT AS #1
20 DEFFNA(X)=(X/256)+128
30 DEFFNB(X)=(X/256)+96
40 LN=0:SC=1:L=600:SP=256
50 X=0:Y=0:ZOOM=1
60 DIM BX(L),BY(L),EX(L),EY(L)
70 BX(0)=X:BY(0)=Y
80 EX(0)=X:BY(0)=Y
90 GOSUB 1200
100 I$=INKEY$
110 IF I$="L" AND F AND L>LN TH
EN LINE (FNA(BX(LN)),FNB(BY(LN)
))-(FNA(EX(LN)),FNB(EY(LN))),15
:LN=LN+1:BX(LN)=X:BY(LN)=Y:EX(L
N)=X:EY(LN)=Y:F=0
120 IF I$="C" THEN X=0:Y=0:EX(L
N)=X:EY(LN)=Y
130 IF I$="M" THEN BX(LN)=X:BY(
LN)=Y
140 IF I$="E" THEN GOSUB 420
150 IF I$="S" THEN GOSUB 600
160 IF I$="D" THEN GOSUB 710
170 IF I$="Z" THEN LV=0:GOSUB 8
40
```



```
180 IF I$=CHR$(29) AND X > -327
68! + SP THEN F=-1:X=X-SP:EX(LN
)=EX(LN)-SP
190 IF I$=CHR$(28) AND X < 3276
8! -SP THEN F=-1:X=X+SP:EX(LN)=
EX(LN)+SP
200 IF I$=CHR$(30) AND Y>-24576
+SP THEN F=-1:Y=Y-SP:EY(LN)=EY(
LN)-SP
210 IF I$=CHR$(31) AND Y<24576-
SP THEN F=-1:Y=Y+SP:EY(LN)=EY(L
N)+SP
220 GOSUB 250
230 GOTO 100
250 BX=FNA(BX(LN)):BY=FNB(BY(LN
))·EX=FNA(EX(LN)):EY=FNB(EY(LN)
)
260 PUT SPRITE 1,(BX-4,BY-4),15
,1
270 PUT SPRITE 2,(EX-4,EY-4),15
,1
280 LINE (10,180)-(255,191),4,B
F
290 PRESET (10,180):PRINT#1,"ES
C:";SC;"     LIN:";LN
300 RETURN
420 SCREEN0
430 INPUT"NOME DE ARQUIVO A SER
 CARREGADO:";F$
440 LN=0·ZOOM=1:SC=1:GOSUB 1200
450 F$="CAS:"+F$
460 OPEN F$ FOR INPUT AS #2
470 INPUT #2,LN
490 FOR U=0 TO LN-1
500 INPUT #2,BX,BY,EX,EY
510 BX(U)=BX:BY(U)=BY:EX(U)=EX:
EY(U)=EY
520 IF ABS(BX(U))<32768! AND AB
S(BY(U))<24576 AND ABS(EX(U))<3
2768! AND ABS(EY(U))<24576 THEN
 LINE(FNA(BX(U)),FNB(BY(U)))  (F
NA(EX(U)),FNB(EY(U))),15
530 NEXT
540 INPUT#2, BX,BY,EX,EY
550 BX(U)=BX:BY(U)=BY:EX(U)=EX:
EY(U)=EY
560 X=EX(LN)·Y=EY(LN)
570 CLOSE #2
590 RETURN
600 SCREEN0
610 INPUT"NOME DE ARQUIVO PARA
GRAVAR:";F$·F$-"CAS·"+F$
620 LV=1 GOSUB 1200·GOSUB 840
630 OPEN F$ FOR OUTPUT AS #2
640 PRINT#2,LN
650 FOR U=0 TO LN
660 PRINT#2,BX(U),BY(U),EX(U),E
Y(U)
670 NEXT
680 CLOSE #2
700 RETURN
710 SCREEN0
720 INPUT"QUER APAGAR QUANTAS L
INHAS",K
730 GOSUB 1200
750 IF K=0 OR LN-K<0 THEN 820
760 LN=LN-K
770 X=BX(LN)·Y=BY(LN'
790 EX(LN)=X·EY(LN)=Y
800 IF LN=0 THEN 830
810 IF ABS(EX(LN-1))<32767 AND
ABS(EY(LN 1))<24576 THEN X=0·Y=
0
820 LV=2·GOSUB 840
830 RETURN
840 IF LN=0 THEN RETURN
850 IF LV-1 THEN ZOOM=1/SC·GOTO
 920
860 IF LV=2 THEN ZOOM=1·GOTO 92
0
870 SCREEN0
```

MICRO DICAS

IDÉIAS PARA O JOGO

Quando você usar o programa apresentado neste artigo como um jogo, a tarefa mais difícil será fazer com que o "tesouro" oculto na figura não apareça como uma mancha muito óbvia para quem o está procurando.

Existem, para isso, vários truques. Um deles consiste em executar um desenho bem elaborado e ocultar o tesouro em um dos detalhes. A dificuldade será ainda maior se você desenhar os detalhes em diferentes graus de ampliação. Isso obrigará seu oponente a também utilizar vários graus de ampliação na procura do tesouro.

Outro truque é a inclusão, no desenho, de pequenos detalhes parecidos com o tesouro, quando se utilizam as reduções. Seu oponente terá que investigar um a um, perdendo precioso tempo com isso.

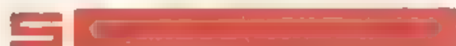
A diversão será maior se você adotar alguma forma de contagem de pontos. Verifique, por exemplo, o número de vezes que o jogador muda a escala de aumento, ou o número de movimentações que ele precisa fazer até achar o tesouro. Outra alternativa é medir o tempo transcorrido entre o início e o fim da busca.

```
880 INPUT "ESCALA DO ZOOM ";ZOO
M
890 GOSUB 1200
900 IF ZOOM=0 THEN 1030
920 FOR U=0 TO LN-1
930 BX(U)  (BX(U)-X)*ZOOM·BY(U)-
(BY(U)  Y)*ZOOM
940 EX(U)=(EX(U)-X)*ZOOM·EY(U)=
(EY(U)  Y)*ZOOM
950 IF ABS(BX(U))<32768! AND AB
S(BY(U))<24576 AND ABS(EX(U))<3
2768! AND ABS(EY(U))<24576 THEN
 LINE(FNA(BX(U)),FNB(BY(U)))-(F
NA(EX(U)),FNB(EY(U))),15
960 NEXT
970 BX(U)=(BX(U)-X)*ZOOM·BY(U)=
(BY(U)-Y)*ZOOM
980 EX(U)=(EX(U)-X)*ZOOM:EY(U)=
(EY(U)-Y)*ZOOM
990 X=EX(LN):Y=EY(LN)
1000 IF ABS(BX(LN))>32767 OR AB
S(BY(LN))>24576 THEN BX(LN)=X:B
Y(LN)=Y
1010 SC=SC*ZOOM
1030 RETURN
1100 DATA 0,0,16,56,16,0,0,0
1200 COLOR 15,4,4:SCREEN 2,0:RE
STORE
1210 FOR I=1 TO 8:READ S:S$=S$+
CHR$(S):NEXT:SPRITE$(1)=S$
1220 RETURN
```

PERIPÉCIAS NO REINO DE NETUNO

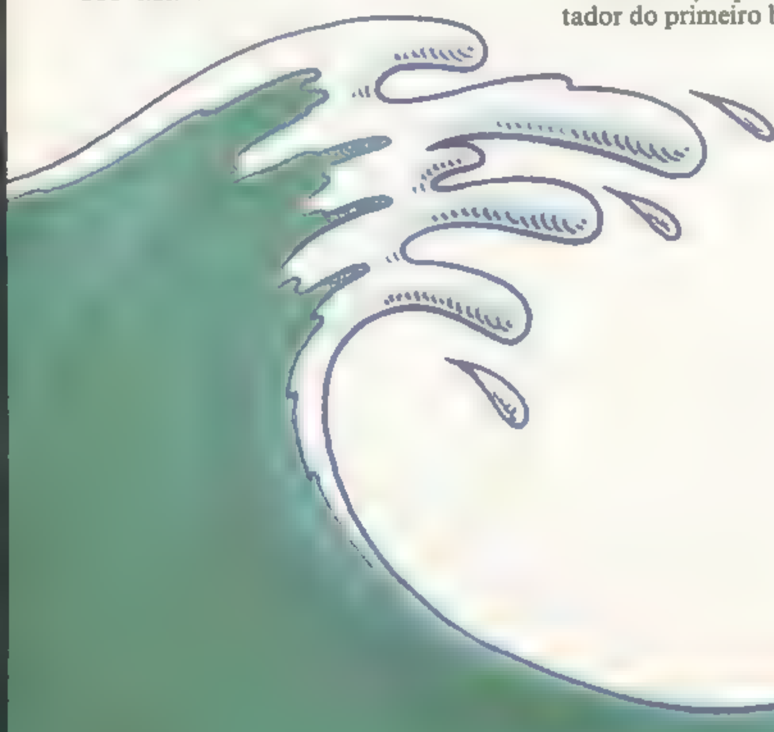
Depois de viver mil aventuras em terra firme, Willie — o infortunado herói do jogo Avalanche — tem que se defrontar com os perigos de um mar agitado. Veja como ele se sai dessa.

Willie está agora num morro a beira-mar. Mas não tem tempo de admirar a paisagem: se ele perder a calma quando as pedras estiverem rolando morro abaixo, ou se tremer com o barulho das cobras, descobrirá tardiamente que está em vias de afogar-se. Para criar o cenário adequado, devemos providenciar vários baldes de água do mar.



A rotina seguinte dá início ao avanço da maré; logo, é melhor que Willie não perca tempo no caminho e retorne rapidamente a seu gostoso piquenique.

```
 10 REM org 58882
 20 REM sea ld bc,57312
 30 REM ld a,(57353)
 40 REM bit 2,a
 50 REM jr z,spt
 60 REM ld bc,57320
 70 REM spt ld hl,(57354)
 80 REM ld a,15
 90 REM ld d,32
100 REM spu push de
110 REM push bc
120 REM call print
130 REM inc hl
140 REM pop bc
150 REM pop de
160 REM dec d
170 REM jr nz,spu
180 REM ld a,(57353)
190 REM dec a
200 REM ld (57353),a
210 REM jr nz,srt
220 REM ld a,10
230 REM ld (57353),a
240 REM ld hl,(57354)
250 REM ld de,32
260 REM sbc hl,de
270 REM ld (57354),hl
280 REM srt ret
290 REM org 58217
300 REM print *
```



Embora pareça que temos na tela um trecho do mar, há na verdade apenas dois caracteres. O primeiro ocupa as oito posições do endereço 57312 em diante, e o segundo as oito posições a partir do endereço 57320. Você pode achar que isso não é água suficiente nem mesmo para molhar os pés de nosso herói. Mas, quando esses dois caracteres forem impressos consecutivamente em linhas alternadas, você construirá rapidamente um oceano.

Como o mar será impresso na tela na proporção de um caractere por vez, recorreremos à rotina **print** de novo. Assim, os parâmetros necessários devem ser carregados nos registros corretos. Como de costume, o par BC carrega o apontador do primeiro byte de dados. O acumulador A carrega o código da cor e o par HL contém a posição da tela onde o dado será impresso. Depois desse lembrete, carregamos o par de registros BC com o endereço do byte inicial que forma o primeiro caractere de mar.

A variável no endereço 57353 é chamada variável de atraso do mar; é ela que controla o movimento das águas. O bit 2 é usado como uma baliza para informar ao processador qual dos dois caracteres de mar deverá ser impresso na última linha.

O conteúdo do atraso do mar é colocado no acumulador e a instrução **bit 2,a** analisa esse bit em particular. Se o seu valor for 0, a instrução **jr z** que vem em seguida faz o processador saltar para a próxima instrução. Mas, se o valor do bit 2 é 1, o salto não ocorre e o par BC é carregado com o endereço do padrão inicial do segundo caractere de mar.

MAR AGITADO

O par de registros HL é carregado com o conteúdo das posições de memória 57354 e 57355. Esses endereços contêm a posição do mar que está sendo impressa no momento e que foi inicializada pela rotina da parte sete de *Avalanche* com a posição do canto inferior esquerdo da tela.

O acumulador A é carregado com o número 15 para dar ao mar a tonalidade de azul adequada. O registro D, por sua vez, é carregado com o número 32; ele será usado como contador para somar 32 colunas através da tela.

O contador em D deve ser preservado intacto. Se o processador utilizar o registro D na rotina **print**, ele se perderá; assim, o par DE é guardado na pilha. Os dados para o caractere de mar serão usados outras vezes, já que cada linha de mar é formada pelo mesmo caractere impresso 32 vezes. Dessa forma, o par BC é guardado na pilha também. A rotina **print** é chamada e os oito bytes do caractere de mar são impressos no lugar correto da tela.

O par HL é então incrementado, movendo o apontador de tela para a próxima posição ao longo da linha. Ao mesmo tempo, o apontador de dados é

- O AVANÇO DA MARÉ
- IMPRESSÃO DOS CARACTERES DO MAR
- TONALIDADE DO MAR
- VARIÁVEL DE ATRASO
- REPETIÇÃO DOS CARACTERES EM LINHAS ALTERNADAS
- A ROTINA **CHARPR**
- MAR BRAVIO

levado de volta para o início dos padrões do caractere (ele tinha sido incrementado durante a rotina **print**), sendo recuperado da pilha. O contador de colunas é também recuperado da pilha e decrementado. Se ele não foi reduzido a zero, a instrução **jr nz** retornará ao início do laço e o processador imprimirá o caractere de mar na posição seguinte da tela ao longo dessa linha.

Quando o contador em D tiver sido reduzido a zero, o processador saltará para fora do laço e prosseguirá com a próxima instrução.

TEMPO E MARÉ

O atraso do mar é carregado sobre o acumulador, decrementado e armazenado de volta em 57353. Se ele não tiver sido reduzido a 0, a instrução **jr nz** fará o processador saltar as instruções restantes até o final da rotina para a qual ele retornará. Se ele tiver sido reduzido a zero, o atraso do mar será ajustado para 10.

O apontador de posição do mar é carregado em HL; 32 é carregado em DE e subtraído de HL; o resultado é armazenado de volta no apontador de localização do mar em 57354. Assim, na próxima vez que a rotina for chamada, o mar terá subido uma linha.



O programa a seguir inicia a expansão da maré.

```
10   ORG 19678
20   SEA LDU #18206
30   LDA 18246
40   BITA #2
50   BEQ SPT
60   LDU #18222
70   SPT LDX 18247
80   LDA #16
90   SPTI PSHS A,U
100   JSR CHARPR
110   PULS U,A
120   DECA
130   BNE SPTI
140   DEC 18246
150   BNE SRT
160   LDA #10
170   STA 18246
180   LDX 18247
190   LEAX -256,X
200   STX 18247
210   SRT RTS
220   CHARPR EQU 19402
```

Para testar esse programa, você precisa carregá-lo no resto de *Avalanche* e executar a seguinte rotina:

```
5 POKE &H467F,&H4C:POKE &H4C80,
&HF3
10 EXEC 19426
20 FOR G=1 TO 160
30 EXEC 19678
40 FOR H=1 TO 100:NEXT H,G
50 GOTO 50
```

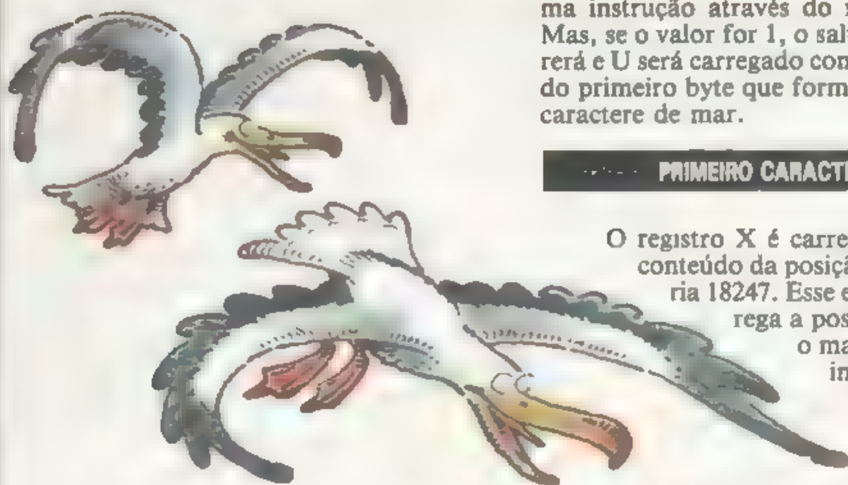
Quando esta rotina estiver funcionando, a maré subirá até que a tela esteja completamente cheia de água. Isso nunca aconteceria durante o jogo, pois Willie teria se afogado antes e o jogo voltaria ao início.

PEQUENAS GOTAS DE ÁGUA

Embora aparentemente tenhamos um trecho do mar representado na tela, existem na verdade apenas dois caracteres importantes na tabela de dados. O primeiro desses caracteres ocupa os oito endereços a partir de 18206 e o segundo ocupa os oito a partir de 18222. Quando esses dois caracteres forem impressos repetidamente em linhas alternadas, você terá um oceano.

Como o mar será impresso na tela do microcomputador, segundo um ritmo de um caractere de cada vez, a rotina **CHARPR** será usada de novo. Logo, seus parâmetros têm de ser carregados nos registros corretos. Como sempre, o registro U carrega o apontador do primeiro byte de dados; então, a própria tabela de dados atua como pilha do usuário. O registro X, por sua vez, car-





ma instrução através do rótulo **SPT**. Mas, se o valor for 1, o salto não ocorrerá e U será carregado com o endereço do primeiro byte que forma o segundo caractere de mar.

PRIMEIRO CARACTERE

O registro X é carregado com o conteúdo da posição de memória 18247. Esse endereço carrega a posição em que o mar está sendo impresso no momento em que foi inicializada pela rotina da parte sete de *Avalanche* no canto inferior esquerdo da tela. O acumulador A é carregado com 16, e será usado como contador para fazer com que dezesseis caracteres de mar sejam impressos.

Esse contador precisa ser preservado intacto enquanto o processador executa a rotina **CHARPR**; portanto, guardamos A na pilha do usuário. Os dados para o caractere de mar serão usados várias vezes, porque cada linha do mar é feita do mesmo caractere — assim, o registro U também é guardado na pilha. A rotina **CHARPR** é chamada e os oito bytes do caractere de mar são impressos no lugar correto da tela.

rega a posição da tela onde o dado será impresso. U é então carregado com o endereço do primeiro byte que forma o primeiro caractere de mar.

A variável no endereço 18246 é chamada variável de atraso do mar e controla o movimento da maré. O bit 2 dessa variável é usado como uma baliza para informar ao processador qual caractere de mar será usado na linha que está sendo impressa.

O conteúdo do atraso do mar é carregado no acumulador e a instrução **BITA #2** analisa esse bit em particular. Se o valor do bit não for 1, a instrução **BEQ** fará o processador pular a próxi-

VEJA O MAR

A rotina **CHARPR** incrementa automaticamente o registro X, de modo que ele esteja pronto para imprimir o próximo caractere ao longo da linha. O apontador de dados é movido de volta ao início dos bytes que formam o caractere de mar por meio de sua recuperação da pilha; esse apontador foi incrementado durante a rotina **CHARPR**. O contador também é recuperado da pilha.

O contador é então decrementado e, se ainda não for zero, a instrução **BNE** saltará para o laço onde o processador imprime o caractere de mar na próxima posição ao longo da linha atual.

Quando o contador em A já for zero, o processador sairá do laço e executará a instrução seguinte.

MAR BRAVIO

O próximo passo consiste em decrementar o atraso do mar. Se esse atraso não tiver sido reduzido a zero, a instrução **BNE** fará o processador saltar as próximas instruções até o final da rotina, para onde ele retornará.

Se ele tiver sido reduzido a zero, o atraso do mar será reajustado com valor 10. O apontador de posição será então carregado no registro X e 256 subtraído dele. O resultado é armazenado

de volta nos endereços que guardam esse apontador, ou seja, 18247. Assim, da próxima vez que essa rotina for chamada, o mar subirá uma linha.



A rotina que vem a seguir inicia a expansão da maré; por isso, é melhor Willie andar o mais rápido possível e recuperar o seu lanche.

```
org 54218
sea ld b,72
ld a,(-5213)
bit 2,a
jr z,spt
ld b,76
spt ld hl,(62407)
ld de,(-5212)
add hl,de
 ld a,b
 ld bc,32
 call 86
 ld a,(-5213)
 dec a
 ld (-5213),a
 jr nz,srt
 ld a,10
 ld (-5213),a
 ld hl,(-5212)
 ld de,32
 sbc hl,de
 ld (-5212),hl
 srt ret
 end
```

Embora tenhamos a impressão de que há um trecho do mar na nossa tela, na verdade só existem dois padrões diferentes. O primeiro tem o código 72 na tabela de padrões e o segundo é o padrão de código 76. À primeira vista, a quantidade de água não é suficiente nem mesmo para molhar os pés de Willie. Mas, quando esses dois padrões forem impressos consecutivamente em linhas alternadas, em pouco tempo teremos, na tela, um oceano.

Como o mesmo padrão de mar será impresso 32 vezes na mesma linha, ocupando-a completamente, usaremos a rotina **86** da ROM. Essa rotina escreve um número na VRAM a partir de um endereço e um certo número de vezes.

Neste programa, a rotina 86 da ROM será usada para colocar o código de um padrão de mar várias vezes na tabela de nomes da VRAM; isso equivale a imprimir o mesmo caractere diversas vezes em sequência, já que a tela é um reflexo da tabela de nomes.

Os parâmetros necessários devem ser carregados nos registros corretos, antes de se chamar a rotina **86**. É preciso também que o acumulador A contenha o código do padrão que vai ser impresso. O par HL, por sua vez, contém a posição da tabela de nomes — ou seja, da tela, onde o padrão começará a ser impresso e o par de registros BC carrega o número de vezes que ele será impresso. Feito o lembrete, continuemos a examinar o programa. O código do padrão do primeiro caractere de mar é colocado no registro B.

A variável no endereço −5213 é chamada variável de atraso do mar; é ela que controla a subida da maré. O bit 2 dessa variável é usado como baliza para indicar ao processador qual dos dois caracteres de mar deve ser impresso na linha atual.

O conteúdo do atraso do mar é colocado no acumulador e a instrução **bit 2,a** analisa esse bit isoladamente. Se o seu valor for 0, a instrução **jr z** que vem em seguida fará o processador saltar a próxima instrução. Mas, se o valor do bit 2 for 1, o salto não será realizado e o registro B será carregado com o código 76 do padrão do segundo caractere de mar.

MAR AGITADO

O par HL é carregado com o endereço inicial da tabela de nomes da VRAM.

que está armazenado nas posições 62407 e 62408. O par DE, por sua vez, é carregado com o valor contido nas posições −5212 e −5211; esse número corresponde à posição na tela do começo da linha que está sendo impressa atualmente e foi inicializado pela rotina da parte sete de *Avalanche* para indicar o canto inferior esquerdo da tela. Os conteúdos de HL e DE são somados em HL, que passa a abrigar o endereço correspondente na tabela de nomes.

O acumulador A recebe então o código do padrão que até agora estava em B. Ao mesmo tempo, o par BC é carregado com o número de vezes em que o padrão deve ser impresso na tela — ou seja, 32, que corresponde ao número de colunas na linha. A seguir, é chamada a rotina **86** da ROM; nesse momento, a linha de mar já estará impressa.

TEMPO DE SUBIDA

O atraso do mar é carregado no acumulador, decrementado e armazenado de volta no endereço −5213. Se o seu valor ainda não tiver sido reduzido a zero, a instrução **jr nz** fará com que o processador salte as instruções restantes até o final da rotina, para a qual ele retornará.

Por outro lado, se ele tiver sido reduzido a zero, o atraso do mar será ajustado para 10.

O apontador de posição do mar é carregado em HL e o número 32 é subtraído dele por meio do par DE; o resultado é armazenado de volta no apontador de posição em −5212 e −5211. Assim, na próxima vez que a rotina for chamada, o mar terá subido uma linha.

Como você já deve ter notado, o atraso do mar controla o tempo que a rotina passará imprimindo a primeira linha; esgotado esse tempo, a rotina é ajustada para subir uma linha a cada dez chamadas.

Depois de jogar algumas vezes, talvez você queira reforçar a dose de emoção de *Avalanche*.

Torne a aventura mais difícil alternando a variável de atraso.



MATEMÁTICA DO CRESCIMENTO

- COMO AS COISAS CRESCEM
- SUPERFÍCIE E VOLUME
- PROBLEMAS ESTRUTURAIS
- RAZÃO DE CRESCIMENTO
- NÚMEROS DE FIBONACCI

Os computadores têm sido utilizados no estudo dos mais diversos aspectos da natureza. Veja aqui como eles podem nos ajudar a compreender o fenômeno do crescimento.

O emprego de computadores na análise das situações relacionadas à nossa realidade não se limita às coisas inanimadas. Ao contrário, a máquina tem se mostrado de grande utilidade no estudo do comportamento dos organismos vivos. Ocorre que a matemática está intimamente associada a natureza — nas formas mais inesperadas. E, sempre que há uma conexão com a matemática, os computadores podem ajudar a compreender o que se passa, efetuando cálculos e indicando resultados.

Neste artigo, examinaremos algumas maneiras de se utilizar computadores na análise do crescimento dos organismos. Todas as formas vivas são capazes, em algum estágio do seu desenvolvimento, de mudar de tamanho, número ou forma. Apenas as duas primeiras situações serão tratadas aqui, juntamente com outros exemplos de inter-relação entre a natureza e a matemática.

COMO AS COISAS CRESCEM

O crescimento — mudança de tamanho — envolve a formação de novos materiais de estrutura. Tanto plantas quanto animais obtêm a matéria-prima necessária para isso do seu meio ambiente. Mas o crescimento geral de um ser pode se dar de duas maneiras: pela multiplicação do número de células (por meio da divisão celular) ou pelo aumento do tamanho das células.

MEDIDAS

Tendo em vista essa diferença, resta saber como medir o crescimento. É melhor medir o peso, como se faz com os bebês, ou a altura, como se faz com as crianças maiores? Ou seria o volume a medida mais adequada? Ou a superfí-



cie do corpo? Não há uma resposta única para tais questões, pois seres diferentes exigem tipos de medida diferentes. Porém, sempre é interessante comparar medidas diversas quando se estuda o crescimento.

As medidas mais significativas para animais são as de volume (ou peso) e superfície corporal. Essas medidas não aumentam no mesmo ritmo, enquanto o animal cresce, o que tem conseqüências importantes, já que o tamanho e a forma de um ser estão muito ligados a seu modo de viver.

Como vimos no artigo da página 434, a área é uma medida quadrática e o volume, uma medida cúbica — isso explica a diferença de velocidade que se registra no aumento dessas duas medidas. A relação entre ambas poderá ser melhor compreendida se tomarmos como exemplo uma forma regular como um cubo. O programa a seguir mostra o que acontece com o volume e a área de cubos que tenham diferentes tamanhos. Digite-o e execute-o.

S

```
10 GOSUB 180
20 LET X=45· LET Y=35
30 LET S=2
40 GOSUB 140
```



```
50 PRINT AT 18.S/5+X/8;S
60 LET SA=6*(S^2)· LET VO=S^3
· LET A$=STR$ (SA/VO)· IF LEN
A$>3 THEN  LET A$=A$( TO 3)
70 PRINT AT 20.S/5+X/8-1;
PAPER 0: INK 7;A$;":1"
80 INPUT "TAMANHO DO CUBO (MA
X 20)  ?".A: IF A<S THEN
 GOSUB 180
 90 LET S=A
100 IF S<1 OR S>20 THEN  GOTO
 80
110 LET X=X+S*5*1.5+5
120 IF X+S*5*1.5>255 THEN
 GOSUB 180  LET X=45
130 GOTO 40
140 PLOT X,Y
150 LET D=S*5
160 DRAW 0,D· DRAW D,0: DRAW D
/3,D/3· DRAW -D,0: DRAW -D/3,-
D/3  DRAW 0,-D· DRAW D,0: DRAW
D/3,D/3: DRAW 0,D: DRAW -D/3,-
D/3: DRAW 0,-D
170 RETURN
180 BORDER 0· PAPER 4· INK 0
 CLS
190 FOR N=0 TO 8: PRINT AT N,0
; PAPER 1;"
                ": NEXT N
200 PRINT AT 18,0; INK 1;"LADO
·";AT 20,0;"A/V:"
210 PRINT AT 0,3; PAPER 1: INK
7;"A/V=PROPORCAO  AREA:VOLUME"
220 RETURN
```







```
10 GOSUB 180:CLS
20 X=45:Y=156
30 S=2
40 GOSUB 140
60 SA=6*S*S:VO=S*S*S
70 PRINT"TAMANHO=";S;TAB(12);"P
ROP.AREA/VOL=";:PRINT USING"#.#
#.1";SA/VO.PRINT
80 INPUT"DIGITE O TAMANHO DO CU
BO (1-20)";A.IF A<S GOSUB 180
90 S=A
100 IF A<I OR S>20 THEN 80
110 X=X+S*7.5+5
120 IF X+S*7.5>255 GOSUB 180:X=
45
130 GOTO 40
140 SCREEN 1,0
150 DRAW"BM"+STR$(INT(X))+","+S
TR$(INT(Y))+"S"+STR$(INT(S*3))+
"C1E2NR6U6C4R6G2L6NE2D6R6NU6E2U
6"
160 IF INKEY$<>" " THEN 160
170 RETURN
180 PMODE 3:PCLS2
190 RETURN
```



```
10 GOSUB 180·CLS
20 X=45:Y=156
30 S=2
40 GOSUB 140
60 SCREEN0·SA=6*S*S:VO=S*S*S
70 PRINT"TAMANHO=";S;TAB(15);"S
UP/VOL=";:PRINTUSING"#.##·1";SA
/VO·PRINT
80 INPUT"TAMANHO DO CUBO? (1-20
)";A:IF A<S THEN GOSUB 180
90 S=A
100 IF A<1 OR A>20 THEN 80
110 X=X+S*7 5+5
120 IF X+S*7.5>255 THEN GOSUB 1
80·X=45
130 GOTO 40
140 SCREEN2
150 LINE (X,Y)-(X+5*S,Y-5*S),8,
B
160 X=X+2*S·Y=Y-2*S:LINE (X,Y)-
(X+5*S,Y-5*S),8,B
165 LINE (X,Y)-(X-2*S,Y+2*S),8:
LINE (X+5*S,Y)-(X+3*S,Y+2*S),8·
LINE (X+5*S,Y-5*S)-(X+3*S,Y-3*S
),8:LINE (X,Y-5*S)-(X-2*S,Y-3*S
),8·Y=156
170 IF INKEY$="" THEN 170 ELSE
RETURN
180 SCREEN2
190 RETURN
```



```
10  HOME : GOSUB 180
20 X = 45:Y = 156
30 S = 2
40  GOSUB 140
60 SA = 6 * S * S:VO = S * S *
S
70  VTAB 21: CALL  - 958· PRINT
 "TAMANHO = ";S: PRINT "RAZAO S
UP/VOL = ";SA / VO;":1"
80  INPUT "TAMANHO DO CUBO (1-2
0)? ";A· IF A < S THEN  GOSUB 1
80
90 S = A
100  IF A < 1 OR S > 20 THEN 80

110 X = X + S * 7.5 + 5
120  IF X + S * 7.5 > 255 THEN
 GOSUB 180:X = 45
130  GOTO 40
140  HPLOT X,Y TO X + 5 * S,Y T
O X + S * 5,Y - S * 5 TO X,Y -
5 * S TO X,Y
150 U =  ABS (U - 1): IF U THEN
 X = X + 2 * S·Y = Y - 2 * S: G
OTO 140
160  HPLOT X,Y TO X - 2 * S,Y +
 2 * S: HPLOT X + 5 * S,Y TO X
+ 3 * S,Y + 2 * S· HPLOT X + 5
* S,Y - 5 * S TO X + 3 * S,Y -
3 * S: HPLOT X,Y - 5 * S TO X -
 2 * S,Y - 3 * S
170 Y = 156· RETURN
180  HGR . HCOLOR= 3: RETURN
```

O programa pede ao usuário que escolha um número para a aresta (a medida de uma quina à outra) do cubo. Em seguida, desenha a figura na tela e mostra a razão entre área e volume. Comece com um cubo pequeno — quatro unidades de aresta, por exemplo — e vá aumentando o valor fornecido ao computador para simular um crescimento. Você verá que, à medida que o cubo se torna maior, a razão entre área e volume diminui. Em outras palavras, o volume aumenta muito mais rapidamente que a superfície do corpo.

Grande parte da listagem está destinada a formatar a tela e desenhar os cubos. Assim, vamos nos deter na linha 60, cujo conteúdo nos interessa de maneira mais direta.

S é a aresta do cubo. A área de um lado é, portanto, S*S e a superfície do cubo todo (ele tem seis lados) é 6*S*S. O volume é S ao cubo, ou S*S*S. A razão entre as duas medidas corresponde ao resultado da divisão da superfície pelo volume. Por exemplo, quando S é igual a 2, a razão é 6*2*2 dividido por 2*2*2, que dá 3:1. Se dobramos o tamanho da aresta, temos 6*4*4 dividido por 4*4*4, que dá 1,5:1 — ou seja, a área é relativamente menor.

Como animais não têm forma de cubo, é provável que queira adaptar o programa para torná-lo mais próximo da realidade. Pequenos animais, como os camundongos, são melhor representados por uma esfera, enquanto seres humanos lembram mais um conjunto de cilindros — um para o tronco e outro para cada membro. No primeiro caso, precisamos conhecer a superfície e o volume da esfera. A superfície é **4*PI*raio^2** e o volume, **4/3*PI*raio^3**. Já a área de um cilindro é **2*PI*raio*altura** mais **2*PI*raio^2** e o volume, **PI*raio^2*altura**.

Seja qual for a forma escolhida, o princípio geral é o mesmo e, em todos os casos, o volume cresce mais rapidamente que a superfície.

CRESCIMENTO E VIDA

Essas relações matemáticas são muito importantes no Reino Animal, afetando as atividades e o habitat do ser em questão. Tomemos como exemplo o rato: ele tem um volume pequeno e, portanto, uma área relativa grande. Isso significa que seu corpo irradia calor rapidamente (dependendo do ambiente em que se encontra). O rato consegue manter-se aquecido produzindo energia pela queima de alimento. Ora, graças à velocidade com que perde calor, ele precisa ingerir alimentos em quantidade equivalente à metade do seu peso.

Um elefante, em contrapartida, tem um volume muito grande em comparação com sua superfície corporal. Portanto, mantidas as proporções, ele irradia menos calor que o rato, e sobrevive com uma quantidade menor (em relação ao seu peso) de alimentos diários. Isso explica por que animais grandes se adaptam melhor ao clima frio dos pólos, onde a comida é escassa.

LIMITE DE TAMANHO

A relação entre massa e superfície ajuda a explicar por que animais e plantas não crescem além de certo tamanho. Dadas as proporções de uma determinada espécie, há um limite definido para as dimensões que ela pode alcançar. Esse limite é imposto pelo peso que os ossos do animal podem suportar.

Quando seu tamanho — isto é, altura, comprimento e largura — dobra, o peso cresce oito vezes, mas a área de suporte dos ossos cresce apenas quatro vezes. Se os ossos se desenvolvessem o suficiente para sustentar o novo peso, ficariam desproporcionalmente finos em relação ao tamanho do animal. Em determinado ponto do seu crescimento, o animal estaria tão desajeitado que não conseguiria se mover.



Volumes e superfícies: gráfico comparativo.



Alterações da taxa de crescimento de uma planta.

Se um rato atingisse o tamanho de um elefante, mantidas as mesmas proporções iniciais, suas pernas seriam incapazes de suportá-lo. Isso explica por que espécies de tamanhos diversos têm proporções tão diferentes.

O maior animal terrestre — o elefante — chega a pesar cerca de 10 toneladas. Um dinossauro podia alcançar 80 toneladas, mas ficava a maior parte do tempo imerso na água. Os animais marinhos contam com o suporte da água, o que amplia os limites de seu tamanho. Baleias azuis têm peso superior a 150 toneladas e medem mais de 30 metros de comprimento. Naturalmente, não haveria pernas que fossem capazes de sustentá-las fora da água. O limite de seu crescimento, assim como dos demais animais marinhos, está mais relacionado à perda de calor.

RAZÃO DE CRESCIMENTO

O programa a seguir mostra a velocidade de crescimento de uma planta, do plantio à maturidade, quando ela chega a seu limite de tamanho.

A planta cresce devagar no início, e rapidamente depois. Atingindo seu limite de tamanho, volta a se desenvolver bem devagar, até parar ou, eventualmente, morrer. Digite e execute o seguinte programa:

S

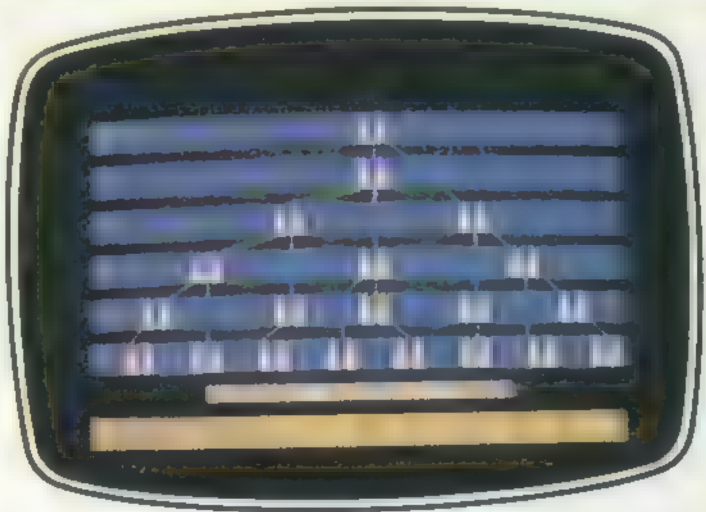
```
10 DIM G(11)
20 DATA 2,9,22,35,58,92,104,112,115,117,118
30 FOR N=1 TO 11: READ G(N): NEXT N
40 BORDER 0: PAPER 0: INK 4: CLS
50 LET X=60: LET Y=10
60 DRAW 80,0
70 PLOT 30,0: DRAW  INK 7;0,168
80 PRINT  INK 2;AT 0,1;"ALTURA"
90 FOR N=0 TO 138 STEP 12
100 PLOT  INK 7;30,N
110 DRAW  INK 7;-5,0
120 NEXT N
130 PRINT AT 20,0;"2";AT 17,0;"6";AT 14,0;"10";AT 11,0;"14";AT 8,0;"20",AT 5,0;"24"
140 PLOT 160,0: DRAW  INK 7;0,168
150 PLOT 160,0: DRAW  INK 7;95,0
160 PRINT AT 0,17; INK 6;"TAXA DE";AT 1,15;"CRESCIMENTO"
170 LET C=1
180 LET GX=161
190 LET GY=1
200 FOR N=1 TO 117
210 IF G(C)<>N THEN  GOTO 250
220 PLOT X,Y: DRAW 9,9,PI/2: DRAW -9,-9,PI/2
230 DRAW -9,9,PI/2: DRAW 9,-9,PI/2
240 LET GX=GX+8: LET GY=1: LET C=C+1
250 PLOT X,Y
260 FOR K=0 TO 3
270 PLOT  INK 5;GX,GY+K: DRAW INK 5;8,0
280 NEXT K
290 LET GY=GY+4
300 LET Y=Y+1
310 NEXT N
320 FOR Y=117 TO 140
330 PLOT X,Y
340 NEXT Y
350 FOR R=1 TO 10 STEP .3
360 CIRCLE  INK 6;X,Y,R
370 NEXT R
380 GOTO 380
```



```
20 DATA 2,9,22,35,58,92,104,112,115,117,118
30 FOR N=0 TO 10:READ G(N):NEXT
40 PMODE 3:PCLS:SCREEN 1,1
50 X=60:Y=190
60 LINE (8,23)-(8,192),PSET:LINE-(80,192),PSET
80 DRAW"BM20,6S24C7D2BRUNLUBR2LDNRDRBRU2BR2LD2RUS8NLS24BED2BRUNLUBRS16RND3RC8"
90 FOR N=47 TO 191 STEP 12
100 LINE (2,N)-(8,N),PSET
120 NEXT
140 LINE (158,23)-(158,191),PSET
150 LINE -(255,191),PSET
160 DRAW"BM176,6C6S24LD2RUS8NLS24BEND2RDLFBRNU2RU2LBR2D2EFU2BRS16RND3RS24BRD2BRUNLUBM182,24ND2RDLFBRU2RDNLDS16BR2U3LR2S24BR2LDNRDR"
170 GX=161:GY=190
190 COLOR 6,7
200 FOR N=1 TO 117
210 IF G(C)>N THEN 250
220 CIRCLE(X,Y-4),15,6,.4,0,.5
230 CIRCLE(X-16,Y),16,6,.4,.75,1:CIRCLE(X+16,Y),16,6,.4,.5,.75
240 GX=GX+8:GY=190:C=C+1
250 PSET(X,Y,6)
260 FOR K=0 TO 3
270 LINE(GX,GY-K)-(GX+8,GY-K),PRESET
280 NEXT
290 GY=GY-4
300 Y=Y-1
310 NEXT
320 FOR Y=75 TO 58 STEP -1
330 PSET (X,Y,6)
340 NEXT:POKE 178,54
350 FOR R=1 TO 15
360 CIRCLE (X,Y),R,.8
370 NEXT
380 GOTO 380
```



Regra de Fibonacci para um retângulo bem proporcionado.



Uma explosão populacional de coelhos.



```
20 DATA 2,9,22,35,58,92,104,112
,115,117,118
30 FOR N=1 TO 10:READ G(N):NEXT
40 COLOR 2,15,15·SCREEN2
50 X=60:Y=190·PI=3.1416
60 LINE (8,23)-(8,192):LINE -(8
0,192)
90 FOR N=47 TO 191 STEP 12
100 LINE (2,N)-(8,N)
120 NEXT
140 LINE (158,23)-(158,191),8
150 LINE -(255,191),8
170 GX=161:GY=190
200 FOR N=1 TO 117
210 IF G(C)>N THEN 250
220 LINE(X,Y)-(X+10,Y-5):LINE(X
,Y)-(X-10,Y-5)
240 GX=GX+8:GY=190:C=C+1
250 PSET(X,Y)
260 FOR K=1 TO 3
270 LINE(GX,GY-K)-(GX+8,GY-K),8
280 NEXT
290 GY=GY-4
300 Y=Y-1
310 NEXT
320 FOR Y=75 TO 58 STEP-1
330 PSET(X,Y),6
340 NEXT
350 FOR R=1 TO 14
360 CIRCLE(X,Y),R,16-R,,,.8
370 NEXT
380 GOTO 380
```



```
20  DATA  2,9,22,35,58,92,104,1
12,115,117,118
30  FOR N = 1 TO 10: READ G(N):
 NEXT
40  HOME : HGR : HCOLOR= 3
50 X = 60:Y = 156
60  HPLOT 8,2 TO 8,158 TO 80,15
8
80  VTAB 21: PRINT "  TAM DA PL
ANTA","    VEL DE CRESCIMENTO"
90  FOR N = 11 TO 155 STEP 12
100  HPLOT 2,N TO 8,N
120  NEXT
140  HPLOT 158,2 TO 158,156 TO
255,156
170 GX = 161:GY = 156
200  FOR N = 1 TO 117
210  IF G(C) > N THEN 250
230  HPLOT X,Y TO X + 10,Y - 5:
 HPLOT X,Y TO X - 10,Y - 5
240 GX = GX + 8:GY = 156:C = C
+ 1
250  HPLOT X,Y
260  FOR K = 1 TO 3
270  HPLOT GX,GY - K TO GX + 8,
GY - K
280  NEXT
290 GY = GY - 4
300 Y = Y - 1
310  NEXT
340 CX = X·CY = Y - 15·R1 = 13:
R2 = 8· HPLOT CX + R,CY
350  FOR A = 0 TO 6.28 STEP .1
360  HPLOT  TO CX + R1 *  COS (
A),CY - R1 *  SIN (A): HPLOT  T
O CX + R2 *  COS (A),CY - R2 *
 SIN (A)
370  NEXT
```

O programa mostra graficamente o que acontece com a planta. Os dados da linha 20 equivalem ao tamanho da planta, medido a intervalos regulares. São usados tanto no desenho da figura quanto no traçado do gráfico que mostra a velocidade de crescimento.

A rotina da linha 200 à 310 encarrega-se do desenho da planta. Os eixos e escalas do gráfico são definidos nas linhas 40 a 160, enquanto o traçado das barras é feito pelas linhas 260 a 280. A flor, finalmente, é desenhada pelas linhas 350 a 380.

Como o gráfico deixa claro, o crescimento da planta é lento no princípio, acelerando em seguida, até perder, de novo, a velocidade. Os dados foram retirados de um experimento real que mediu o crescimento da área de uma folha de um pepineiro. Os mesmos valores podem ser usados para avaliar o crescimento de toda a planta.

GERAÇÕES E NÚMEROS

Quando se acasalam, os animais multiplicam-se, dando origem a várias gerações. Tomemos o caso dos coelhos. Um par de coelhos — primeira geração — produz um novo par. Este forma a segunda geração. O casal inicial produz mais um par e a segunda geração passa a ter dois pares. Como veremos adiante, uma série de números representando gerações pode ser como se segue: 1,1,2,3,5,8,13,21 etc. O próximo programa mostra graficamente como o número de coelhos cresce com o passar do tempo.



```
10 BORDER 0: PAPER 1: INK 7:
CLS
20 FOR N=0 TO 7: READ A: POKE
USR "a"+N,A: NEXT N
30 FOR N=0 TO 7: READ A: POKE
USR "b"+N,A: NEXT N
40 LET C$="


50 LET A$=CHR$ 144: LET B$=
CHR$ 145
60 PAPER 0: CLS : PAPER 1
70 FOR N=1 TO 6: PRINT C$'':
NEXT N
80 PRINT  INK 5;AT 0,0;"GER"
```

```
-''"0"''"1"''"2"''"3"''"4"''"5"
 90 FOR N=1 TO 20
100 IF N>1 THEN  FOR P=1 TO 10
 : SOUND .01,P: NEXT P
110 READ X,Y: PRINT AT Y,X;A$;
A$,AT Y+1,X;B$;B$
120 NEXT N
130 FOR N=1 TO 20
140 READ X,Y,XX,YY: PLOT X,Y:
DRAW  INK 4;XX,YY
150 NEXT N
160 PRINT AT 18,2; INVERSE 1;"
ARVORE GENEALOGICA - COELHOS"
170 INK 6: INVERSE 1: PRINT '"
GERACAO          : 0  1  2  3  4
 5"''"PARES          : 1  1  2
3  5  8"
180 GOTO 180
190 DATA 144,80,48,28,52,62,62
,24,60,126,118,120,126,254,252
,63
200 DATA 16,0,16,3,22,6,11,6,
16,9,6,9,11,12,25,9,22,12,3,12
,6,15,14,15,16,12,28,12,26,15,
2,15,10,15,18,15,22,15,30,15
210 DATA 136,159,0,-7,144,159,
32,-31,128,135,-34,-7,136,135,
0,-31,184,112,0,-31,192,112,8,
-8,88,111,-25,-7,96,111,0,-31
220 DATA 48,87,-10,-7,56,87,0,
-31,24,63,-5,-7,90,63,-8,-7,
128,87,-8,-8,120,79,0,-23,136,
87,0,-7
230 DATA 144,63,7,-7,184,63,0,
-7,208,88,0,-31,216,88,7,-7,
240,63,7,-7
```



```
10 PMODE 3:PCLS:DIM R(9)
20 SS=PEEK(186)*256+PEEK(187)
30 FOR K=SS TO SS+480 STEP 32
40 READ A,B:POKE K,A:POKE K+1,B
50 NEXT
60 GET (2,0)-(13,15),R,G
70 PUT(14,0)-(25,15),R,PSET:GET
(2,0)-(25,15),R,G
80 PCLS4:SCREEN 1,0
90 COLOR3: FOR K=0 TO 5
100 LINE (0,32*K)-(255,32*K+24)
,PSET,BF
110 NEXT
120 COLOR 1:FOR N=1 TO 20
140 READ X,Y:PUT(X,Y)-(X+23,Y+1
5),R,PSET
150 IF N>1 THEN PLAY"O1T50CCEFG
AB"
160 READ X,Y,XX,YY:LINE(X,Y)-(X
X,YY),PSET
170 NEXT
180 GOTO 180
190 DATA 169,170,153,170,165,17
0,169,90,165,154,165,86,15,90,1
69,106
200 DATA 169,106,165,90,165,154
,165,106,165,106,149,106,149,10
6,165,90
210 DATA 114,7,124,24,124,38,11
4,39,113,56,69,70,59,71,138,23,
178,68,171,71,57,89,42,100,31,1
03
220 DATA 124,57,124,101,115,103
,195,87,208,101,199,103
230 DATA 29,120,12,133,3,135,68
,89,68,133,59,135,124,121,124,1
33,115,135
240 DATA 180,89,180,133,171,135
,222,120,234,133,225,135,12,153
,12,165,3,167
250 DATA 40,121,42,165,33,167,6
8,153,72,165,63,167,113,120,98,
165,93,167,124,153,144,165,137,
167
260 DATA 178,153,176,165,167,16
7,208,121,206,165,197,167,234,1
53,234,165,225,167,124,70,124,7
0
```



```
5 CLEAR 5000
10 COLOR 4,15,15:SCREEN2
60 A$="S4F1D1L2BF1R3G1L1D1R1F1L
3D1R3"
90 FOR K=0 TO 5
100 LINE(0,32*K)-(255,32*K+24),
2,BF
110 NEXT
120 FOR N=1 TO 20
140 READ X,Y:PRESET(X,Y):DRAW"X
A$;":PRESET(X+15,Y):DRAW"XA$;"
160 READ X,Y,XX,YY:LINE(X,Y)-(X
X,YY),8
170 NEXT
180 GOTO 180
210 DATA 114,7,124,24,124,38,11
4,39,113,56,69,70,59,71,138,23,
178,68,171,71,57,89,42,100,31,1
03
220 DATA 124,57,124,101,115,103
,195,87,208,101,199,103
230 DATA 29,120,12,133,3,135,68
,89,68,133,59,135,124,121,124,1
33,115,135
240 DATA 180,89,180,133,171,135
,222,120,234,133,225,135,12,153
,12,165,3,167
250 DATA 40,121,42,165,33,167,6
8,153,72,165,63,167,113,120,98,
165,93,167,124,153,144,165,137,
167
260 DATA 178,153,176,165,167,16
7,208,121,206,165,197,167,234,1
53,234,165,225,167,124,70,124,7
0
```



```
10  HGR2 : HCOLOR= 3
20  DATA  1,0,4,0,60,60,28,109,
1,193,193,43,53,55,53,54,55,45,
53,63,63,46,173,27,54,45,45,62,
63,63,46,45,53,63,55,45,53,63,6
2,63,60,7,0
30 P = 233: POKE P - 1,0: POKE
P,97: SCALE= 1: ROT= 0
40  FOR I = 24832 TO 24874: REA
D C: POKE I,C: NEXT
90  FOR K = 0 TO 5
100  HPLOT 0,30 * K TO 255,30 *
 K
110  NEXT
120  FOR N = 1 TO 20
140  READ X,Y: DRAW 1 AT X,Y: D
RAW 1 AT X + 15,Y
160  READ X,Y,XX,YY: HPLOT X,Y
TO XX,YY
170  NEXT
190  DATA   114,7,124,24,124,38
,114,39,113,56,69,70,59,71,138,
```



```
23,178,68,171,71,57,89,42,100,3
1,103
200  DATA   124,57,124,101,115,1
03,195,87,208,101,199,103
210  DATA    29,120,12,133,3,135
,68,69,68,133,59,135,124,121,12
4,133,115,135
220  DATA    180,89,180,133,171,
135,222,120,234,133,225,135,12,
153,12,165,3,167
230  DATA   40,121,42,165,33,167
,68,153,72,165,63,167,113,120,9
8,165,93,167,124,153,144,165,13
7,167
240  DATA    178,153,176,165,167
,167,208,121,206,165,197,167,23
4,153,234,165,225,167,124,70,12
4,70
```

O programa começa criando o UDG para o coelho, a partir das linhas de dados (190 e 200). As linhas 60 a 110 (40 a 80, no Spectrum) desenham barras na tela para separar cada geração. A seção seguinte do programa, que vai até a linha 180, imprime os coelhos e as linhas que ligam as gerações. Os dados que definem as linhas e posições são lidos da linha 210 em diante.

À medida que os organismos se multiplicam, eles se espalham e colonizam outras áreas. Um bom exemplo é a maneira pela qual as bactérias se reproduzem. Computadores também podem ser usados para representar esse tipo de expansão. Existem jogos interessantes elaborados a partir dessa idéia. Você encontrará um deles, o *Jogo da Vida*, na página 961.

NÚMEROS DE FIBONACCI

A série de números mencionada anteriormente — 1,1,2,3,5,8,13,21 etc. — apresenta algumas propriedades que podem ser observadas na natureza e em obras de arte. É conhecida como "números de Fibonacci" desde o século XIII, quando o matemático italiano a descreveu. Cada número dessa série corresponde à adição dos dois anteriores.

Uma propriedade interessante pode ser observada tomando-se quaisquer três números em seqüência. Multiplique o primeiro pelo último e compare o resultado com o quadrado do número do meio. A diferença será sempre 1. Tomemos os números 5, 8 e 13: 5 vezes 13 dá 65 e 8 ao quadrado, 64.

Dividindo cada número pelo da direita, obtemos uma série de frações que se relacionam à natureza e à arte. Descobriu-se, por exemplo, que nem todas as formas retangulares são igualmente agradáveis de se olhar. Algumas parecem muito estreitas, outras muito largas. O retângulo de melhor aparência tem uma relação especial entre altura e largura, conhecida como a "razão de ouro". Essa razão é igual a (sqr (5) – 1)/2, cujo resultado é 0.6180. Se você calcular qualquer das frações de Fibonacci, verá que, quanto maiores os números usados, mais elas se aproximam da razão de ouro. Por exemplo, 8/13 é igual a 0,6154; 13/21, a 0,6190 e 21/34, a 0,6176.

Experimente agora o próximo programa. Ele desenha retângulos de diferentes tamanhos, de maneira que você mesmo poderá julgar quais os melhores.



```
10 DIM F(12): DIM D(14)
20 LET D(1)=1: LET D(2)=1
30 FOR N=3 TO 14
40 LET D(N)=D(N-1)+D(N-2)
50 NEXT N
60 FOR N=1 TO 12
70 LET F(N)=D(N)/D(N+1)
80 NEXT N
90 BORDER 0: INK 7: PAPER 0·
CLS
100 LET A=15· LET B=8
110 LET X=20: LET Y=170
120 GOSUB 310
130 PLOT 0,130: DRAW  INK 2;
255,0
140 PLOT 0,128· DRAW  INK 2;
255,0
150 PLOT 80,130: DRAW  INK 2;0
,45
160 PLOT 82,130: DRAW  INK 2;0
,45
170 PRINT AT 2,1; INK 3;"B";AT
4,5;"A"
180 INPUT "COMPRIMENTO DO LADO
 A (MAX 70) ?";A
190 IF A<1 OR A>70 THEN  GOTO
 180
200 INPUT "COMPRIMENTO DO LADO
 B (MAX 40) ?";B
210 IF B<1 OR B>40 THEN  GOTO
 200
220 LET X=128-(A*3/2): LET Y=
 120
230 GOSUB 310
240 PRINT AT 0,11; INK 5;"LADO
 A=";A;" LADO B=";B
250 FOR N=1 TO 12
260 IF A/B=F(N) OR B/A=F(N)
 THEN  PRINT AT 2,8;"RAZAO DE
FIBONACCI"
270 NEXT N
280 PRINT AT 4,1; FLASH 1; INK
6;"QUALQUER TECLA PARA CONTINU
AR"
290 PAUSE 0
300 RUN
310 PLOT X,Y: DRAW 3*A,0: DRAW
0,-3*B
320 DRAW -3*A,0: DRAW 0,3*B
330 RETURN
```

```
10 DIM F(11),D(13)
20 D(0)=1:D(1)=1
30 FOR N=2 TO 13
40 D(N)=D(N-1)+D(N-2)
50 NEXT
60 FOR N=0 TO 11
70 F(N)=D(N)/D(N+1)
80 NEXT
90 PMODE 3:PCLS:CLS
100 A=15:B=8
110 X=20:Y=22
120 GOSUB 310
130 COLOR 4:LINE(0,62)-(255,62)
,PSET
140 LINE(0,64)-(255,64),PSET
150 LINE(80,62)-(80,17),PSET
160 LINE(82,62)-(82,17),PSET
170 DRAW"BM9,38C2S8U4R2FGNLFGL2
BM40,58U3EFDNLD2"
175 FOR K=1 TO 900.NEXT
180 INPUT"COMPRIMENTO DO LADO A
 (MAX 70) ";A
190 IF A<1 OR A>70 THEN 180
200 INPUT"COMPRIMENTO DO LADO B
 (MAX 40) ";B
210 IF B<1 OR B>40 THEN 200
220 X=128-A*S/3.Y=72
230 GOSUB 310
250 FOR N=0 TO 11
260 IF A/B=F(N) OR B/A=F(N) THE
N DRAW"BM106,44C2S8NR2D2NRD2BR4
U4BR2ND4RFGNLFGLBR4NU4R2U4L2BR4
DND3F2NU3DBR2U3EFDNL2D2BR4L2U4R
2BR4L2D4R2BR2U4BR8ND4R2D2L2F2BR
2U3EFDNLD2BR3U4LR2BR2D4BR2R2U4L
2D4"
270 NEXT
280 IF INKEY$="" THEN 280
300 RUN
310 SCREEN 1,0:COLOR 3
320 LINE(X,Y)-(3*A+X,Y+3*B),PSE
T,BF
330 RETURN
```

```
10 DIM F(11),D(13)
20 D(0)=1:D(1)=1
30 FOR N=2 TO 13
40 D(N)=D(N-1)+D(N-2)
50 NEXT
60 FOR N=0 TO 11
70 F(N)=D(N)/D(N+1)
80 NEXT:OPEN "GRP:" FOR OUTPUT
AS #1
85 COLOR  15,2,2·GOTO 180
90 SCREEN2
100 A=15·B=8
110 X=20·Y=22
120 GOSUB 310
130 LINE(0,62)-(255,62),8
140 LINE(0,64)-(255,64),8
150 LINE(80,62)-(80,17),8
160 LINE(82,62)-(82,17),8
170 FOR K=1 TO 1500·NEXT·RETURN
180 SCREEN0:INPUT"TAMANHO DO LA
DO A (MAX 70)";AA
190 IF AA<1 OR AA>70 THEN 180
200 INPUT"TAMANHO DO LADO B (MA
X 40)";BB
210 IF BB<1 OR BB>40 THEN 200
215 GOSUB 90
220 A=AA·B=BB·X=128-A*3/2·Y=72
230 GOSUB 310
240 FOR N=0 TO 11
260 IF A/B=F(N) OR B/A=F(N) THE
N PRESET(90,15):PRINT#1,"RAZAO
DE FIBONACCI"
270 NEXT
280 IF INKEY$="" THEN 280
300 RUN
310 REM
320 LINE(X,Y)-(3*A+X,Y+3*B),4,B
F
330 RETURN
```



```
10  DIM F(11),D(13)
20 D(0) = 1:D(1) = 1
30  FOR N = 2 TO 13
40 D(N) = D(N - 1) + D(N - 2)
50  NEXT
60  FOR N = 0 TO 11
70 F(N) = D(N) / D(N + 1)
80  NEXT
90  HOME : HGR : HCOLOR= 3
100 A = 15:B = 8
110 X = 20·Y = 22
120  GOSUB 310
130  HPLOT 0,62 TO 255,62
140  HPLOT 0,64 TO 255,64
150  HPLOT 80,62 TO 80,17
160  HPLOT 82,62 TO 82,17
180  VTAB 21: INPUT "TAMANHO DO
 LADO A (MAX 70) ";A
190  IF A < 1 OR A > 70 THEN 18
0
200  INPUT "TAMANHO DO LADO B (
MAX 39) ";B
210  IF B < 1 OR B > 39 THEN 20
0
220 X = 128 - A * 3 / 2:Y = 72
230  GOSUB 310
250  FOR N = 0 TO 11
260  IF A / B = F(N) OR B / A =
 F(N) THEN  VTAB 21: CALL  - 95
8: PRINT " RAZAO DE FIBONACCI "
: CHR$ (7)
270  NEXT
280  FOR I = 1 TO 1000: NEXT :
POKE  - 16302,0: GET R$
300  GOTO 90
310  FOR XX = X TO X + 3 * A
320  HPLOT XX,Y TO XX,Y + 3 * B

325  NEXT
330  RETURN
```

O programa começa pedindo que o usuário defina o tamanho dos lados do retângulo. Se você entrar dois números adjacentes da série de Fibonacci, será avisado de que se trata de uma fração de Fibonacci.

Os números da série são calculados nas linhas que vão de 20 a 50. A partir dos dois primeiros números, os demais vão sendo calculados pela adição de cada número ao anterior. As linhas 60 a 80 guardam os números em uma matriz e as linhas 90 a 170 desenham um retângulo, para exemplo. Em seguida, começa a rotina de entrada, que verifica se os números escolhidos pertencem à série de Fibonacci.

Podem-se encontrar exemplos dos números de Fibonacci também na natureza; assim, uma espiral ligando folhas de um galho tem voltas e vãos que formam razões de Fibonacci. Conte o número de voltas da espiral, de uma folha à outra. Depois, conte o número de vãos da espiral entre essas duas posições. A razão é, em geral, 5/3 ou 8/5.

JOGOS DE GUERRA: ÀS ARMAS

A batalha começa, afinal, possibilitada pelas rotinas de combate balístico e corpo a corpo que apresentamos neste artigo. Veremos também como testar o moral e contar as baixas de cada lado.

Capa e Espada está quase completo; já podemos dar ordens às unidades e movê-las pelo campo de batalha. Faltam apenas as rotinas de combate. Depois de adicioná-las ao programa, experimentaremos, finalmente, o jogo.

Os eventos resultantes do choque entre dois exércitos podem ser bem complicados — assim, antes de qualquer coisa, devemos decidir o que incluir no programa. No tipo mais simples de resolução de combate, o maior sempre vence, o que faz do tamanho das unidades o fator decisivo da vitória ou da derrota. Podemos ainda relacionar o desenlace da batalha ao moral da tropa, ao número de cavaleiros etc. Na realidade, ninguém tem a receita infalível do triunfo; portanto, os fatores que determinam quais serão os vencedores em um jogo de guerra dependem da escolha do programador.

COMBATE BALÍSTICO

Em *Capa e Espada* existem dois tipos de combate: balístico (flechas) e corpo a corpo. Esta primeira rotina trata do combate com projéteis.



```
1710 REM  Tiro
1720 GOSUB 2540
1730 PRINT AT 18,0;"Unidade ";s
h;" atira"
1740 LET fx=5: LET fy=5: LET gp
=-1
1745 LET st=9
1750 IF sh>8 THEN  LET st=1
1770 FOR m=st TO (st+7)
1780 LET tm=ABS (T(m,8)-T(sh,8)
): LET ty=ABS (T(m,9)-T(sh,9))
1785 IF tm<fx AND T(m,1)<5 AND
ty<fy THEN  LET fx=tm: LET fy=t
y: LET gp=m
1790 NEXT m
1800 IF gp=-1 THEN  PRINT AT 19
,0;"Fora de alcance": GOSUB 241
0: RETURN
1810 LET C=8-T(gp,4)-ABS(fx-fy )
1820 IF gp<3 OR gp=9 OR gp=10 T
HEN  LET C=C+1
1830 IF m(T(gp,8),T(gp,9))=2 TH
EN  LET C=C-2
1840 IF T(gp,1)<>2 THEN  LET C=
C+1
1850 LET C=(C+(INT (T(sh,7)/40)
)+FN r(3))*10
1860 LET T(gp,7)=T(gp,7)-C
1870 PRINT "Houve ";C;" baixas
na unidade ";gp
1875 GOSUB 2410
1880 LET un=gp: GOSUB 2200
1890 RETURN
```

```
1710 REM TIRO
1720 GOSUB 2540
1730 LOCATE 0,19:PRINT "UNIDADE
";SH;"ATIRA"
1740 FX=5:FY=5:GP=-1
1745 ST=9
1750 IF SH>8 THEN ST=1
1770 FOR M=ST TO ST+7
1780 TM=ABS(T(M,8)-(T(SH,8)):TY
=ABS(T(M,9)-T(SH,9))
1785 IF TM<FX AND T(M,1)<5 AND
TY<FY THEN FX=TM:FY=TY:GP=M
1790 NEXT M
1800 IF GP=-1 THEN LOCATE 0,20:
PRINT "FORA DE ALCANCE":GOSUB 2
410
1810 C=8-T(GP,4)-ABS(FX-FY)
1820 IF GP<3 OR GP=9 OR GP=10 T
HEN C=C+1
1830 IF M(T(GP,8),T(GP,9))=2 TH
EN C=C-2
1840 IF T(GP,1)<>2 THEN C=C+1
1850 C=(C+INT(T(SH,7)/40))+FN R
(3))*10
1860 T(GP,7)=T(GP,7)-C
1870 PRINT "HOUVE";C;"BAIXAS NA
 NIDADE";GP
1875 GOSUB 2410
1880 UN=GP:GOSUB 2200
1890 RETURN
```



```
1710  REM  TIRO
1720  GOSUB 2540
1730  VTAB 21: PRINT "UNIDADE "
;SH;" ATIRA": GOSUB 30000
1740 FX = 5:FY = 5:GP =  - 1
1745 ST = 9
1750  IF SH > 8 THEN ST = 1
1770  FOR M = ST TO (ST + 7)
1780 TM =  ABS (T(M,8) - T(SH,8
)):TY =  ABS (T(M,9))
1785  IF TM < FX AND T(M,1) < 5
 AND TY < FY THEN FX = TM:FY =
TY:GP = M
1790  NEXT M
1800  IF GP =  - 1 THEN  PRINT
```

```
"FORA DE ALCANCE": GOSUB 30000: GOSUB 2410: RETURN
1810 C = 8 - T(GP,4) - ABS (FX - FY)
1820 IF GP < 3 OR GP = 9 OR GP = 10 THEN C = C + 1
1830 IF M(T(GP,8),T(GP,9)) = 2 THEN C = C - 2
1840 IF T(GP,1) < > 2 THEN C = C + 1
1850 C = (C + ( INT (T(SH,4) / 40)) + FN R(3)) * 10
1860 T(GP,7) = T(GP,7) - C
1870 PRINT "HOUVE ";C;" BAIXAS NA UNIDADE ";GP
1875 GOSUB 30000: GOSUB 2410
1880 UN = GP: GOSUB 2200
1890 RETURN
```



Modifique as seguintes linhas do programa destinado ao Apple:

```
1730 VTAB 21: PRINT "UNIDADE ";SH;" ATIRA"
1800 IF GP = - 1 THEN PRINT "FORA DE ALCANCE": GOSUB 2410: RETURN
1875 GOSUB 2410
```



```
1710 REM TIRO
1720 GOSUB 2540
1730 DRAW"BM0,152":A$="UNIDADE"+STR$(SH)+" ATIRA" GOSUB 3190
1740 FX=5:FY=5:GP=-1
1745 ST=9
1750 IF SH>8 THEN ST=1
1770 FOR M=ST TO (ST+7)
1780 TM=ABS(T(M,8)-T(SH,8)):TY=ABS(T(M,9)-T(SH,9))
1785 IF TM<FX AND T(M,1)<5 AND TY<FY THEN FX=TM:FY=TY:GP=M
1790 NEXT M
1800 IF GP=-1 THEN DRAW"BM0,160":A$="FORA DE ALCANCE":GOSUB 3190:GOSUB 2410:RETURN
1810 C=8-T(GP,4)-ABS(FX-FY)
1820 IF GP<3 OR GP=9 OR GP=10 THEN C=C+1
1830 IF M(T(GP,8),T(GP,9))=2 THEN C=C-2
1840 IF T(GP,1)<>2 THEN C=C+1
1850 C=(C+INT(T(SH,7)/40))+RND(3))*10
1860 T(GP,7)=T(GP,7)-C
1870 DRAW"BM0,160":A$="HOUVE"+STR$(C)+" BAIXAS NA UNIDADE"+STR$(GP):GOSUB 3190
1875 GOSUB 2410
1880 UN=GP:GOSUB 2200
1890 RETURN
```

A rotina de tiro é chamada sempre que os arqueiros recebem a ordem "FOGO". Em outros jogos pode haver mais de um tipo de unidade com capacidade de atirar diferentes projéteis.

Quando uma unidade recebe a ordem de disparo, **G%** (**GP** ou **gp**) passa a valer –1 na linha 1740. A seguir, a rotina verificará se há um alvo vulnerável na área. As coordenadas da unidade que atira são comparadas às de cada unidade inimiga pela linha 1780. Se o alvo estiver dentro de um alcance de cinco posições do mapa, **G%** (**GP** ou **gp**) assume o número da unidade atingida. Se houver mais de um alvo, o mais próximo será considerado.

Caso não haja unidades inimigas por perto, **G%** (**GP** ou **gp**) permanece valendo –1 e o jogador é informado que o inimigo está "FORA DE ALCANCE".

Se o disparo atingir o alvo, as baixas inimigas serão calculadas e armazenadas em **C%** (ou **C**).

Vários fatores determinam a gravidade dos danos causados por um disparo. Na linha 1810, o tipo de armadura da unidade-alvo é subtraído de 8. Em seguida, subtrai-se do resultado um fator de distância. A linha 1820 adiciona 1 a **C%** (ou **C**), se a unidade-alvo for de cavaleiros. Se ela estava resguardada por uma floresta (terreno tipo 2), a linha 1830 subtrai 2; se se encontrava em movimento, a linha 1840 adiciona 1 (tropas em movimento são mais vulneráveis à artilharia).

Finalmente, a linha 1850 soma o resultado a um quarto do poder da unidade atacante, mais uma parcela aleatória — tudo isso vezes dez. O resultado corresponde ao total de baixas da unidade-alvo. As baixas são subtraídas do poder da unidade e, em seguida, a rotina que testa o moral é chamada.

O CONFRONTO

O resultado do combate corpo a corpo é calculado de maneira similar:



```
1510 REM  Combate
1520 IF (us<9 AND th<9) OR (us>8 AND th>8) THEN  RETURN
1530 IF T(us,1)=5 OR T(th,1)=5 THEN  RETURN
1540 GOSUB 2540
1550 PRINT AT 18,0;"Combate !!"
1560 LET at=INT ((T(us,7)-T(th,7))/50)
1570 LET at=at+T(us,3)-T(th,4)+T(us,5)+FN r(5)
1580 IF ABS (T(us,2)-T(th,2))<>2 THEN  LET at=at+2
1590 IF us<3 OR us=9 OR us=10 THEN  LET at=at+1
1600 LET df=INT ((T(th,7)-T(us,7))/60)
1610 LET df=df+T(th,3)-T(us,4)+T(th,5)+m(T(th,8),T(th,9))+FN r(3)+2
1615 LET wn=th: LET lo=us
1620 IF at>df THEN  LET wn=us: LET lo=th
1630 LET wc=INT (T(wn,7)/10): IF wc<1 THEN  LET wc=1
1640 LET T(wn,7)=T(wn,7)-wc
1650 LET lc=INT (T(lo,7)/5): IF lc<1 THEN  LET lc=1
```

```
1660 LET [illegible]
1670 PRINT [illegible] perde [illegible]
lo;" perde ";l[illegible]
1680 GOSUB 2410
1690 LET [illegible]
1700 RETURN
```

```
1510 REM COMBATE
1520 IF (US<9 AND TH<9) OR [illegible]
9 AND TH>8) THEN RETURN
1530 IF T(US,1)=0 OR T(TH,1)=0
THEN RETURN
1540 GOSUB 2540
1550 LOCATE [illegible] PRINT "COMBATE
!!!"
1560 AT=INT([illegible]
)
1570 AT=AT+T(US,3)+[illegible]
5)+FN R(5)
1580 IF ABS(T(US,2)-T(TH,2))[illegible]
THEN AT=AT+2
1590 IF US<3 OR US=9 OR US=10 T
HEN AT=AT+1
1600 DF=INT([illegible]
)
1610 DF=DF+T(TH,3)-T(US,4)+T(TH
,5)+M(T(TH,8),T(TH,9))+FN R(3)[illegible]
1615 WN=TH:LO=US
1620 IF AT>DF THEN WN=US:LO=TH
1630 WC=INT(T(WN,7)/10):IF WC<1
THEN WC=1
1640 T(WN,[illegible]
1650 LC=INT[illegible]
THEN LC=1
1660 [illegible]
```

```
1670 PRINT WN;" PERDE ";WC;" ";LO;
"PERDE";LC
1680 GOSUB 2410
1690 UN=LO:GOSUB 2200
1700 RETURN
```

```
1510 REM COMBATE
1520 IF (US < 9 AND TH < 9) OR
(US > 8 AND TH > 8) THEN RETU
RN
1530 IF T(US,1) = 0 OR T(TH,1)
= 0 THEN RETURN
1540 GOSUB 2540
1550 VTAB 21: PRINT "COMBATE
!!": GOSUB 30000
1560 TT = INT ((T(US,7) + T(TH
,7)) / 60)
1570 TT = TT + T(US,3) + T(TH,4
) + T(US,5) + FN R(5)
1580 IF ABS (T(US,2) - T(TH,2
)) < > 2 THEN TT = TT + 2
1590 IF US < 3 OR US = 9 OR US
= 10 THEN TT = TT + 1
1600 DF = INT ((T(TH,7) + T(US
,7)) / 60)
1610 DF = DF + T(TH,3) + T(US,4
) + T(TH,5) + M(T(TH,8),T(TH,9))
+ FN R(3) + 1
1615 WN = TH: LO = US
1620 IF TT > DF THEN WN = US: L
O = TH
1630 WC = INT (T(WN,7) / 10):
IF WC < 1 THEN WC = 1
1640 T(WN,7) = T(WN,7) - WC
1650 LC = INT (T(LO,7) / 5): I
F LC < 1 THEN LC = 1
```

```
1660 T(LO,7) = T(LO,7) - LC
1670 PRINT WN;" PERDE ";WC;" ";
LO;" PERDE ";LC: GOSUB [illegible]
1680 GOSUB 2410
1690 UN = LO: GOSUB [illegible]
1700 RETURN
```

[illegible]

```
1550 VTAB 21: PRINT "COMBATE !
!!"
1670 PRINT WN;" PERDE ";WC;" ";
[illegible] PERDE ";LC
```

```
1510 REM COMBATE
1520 IF [illegible]
[illegible] THEN RETURN
1530 IF T(US,[illegible]
THEN RETURN
1540 GOSUB 2540
1550 DRAW [illegible]
";GOSUB 3190
1560 [illegible]
1570 AT=[illegible]
5)+RND(5)
1580 IF ABS([illegible]
THEN AT=AT+2
1590 IF US<[illegible]
HEN AT=AT+1
1600 [illegible]
[illegible]
```



```
[illegible]
1640 [illegible]
1650 LC=INT(T(LO,7)/5):IF LC<1
[illegible]
1660 [illegible]
1670 [illegible]
 PERDE"+STR$(LC):GOSUB 3190
1680 GOSUB 2410
1690 [illegible]
[illegible]
```

Essa rotina é chamada sempre que duas unidades se encontram no tabuleiro. Se elas forem do mesmo exército, a rotina é imediatamente interrompida. O mesmo acontece se uma das unidades estiver em retirada (linha 1530).

A principal diferença entre o combate balístico e o corpo a corpo é que, no segundo caso, as duas unidades entram em ação. O combate corpo a corpo é, portanto, mais complexo, exigindo que se calculem as baixas dos dois lados.

A linha 1560 dá ao atacante uma nota igual a um quinto da diferença entre o poder das duas unidades. Em seguida, a diferença entre a arma do atacante e a armadura do atacado é somada ao valor do moral do atacante mais um número aleatório de 1 a 5.

Aquele que ataca tem [illegible] adicional se o inimigo não [illegible] que se movia em direção diferente da do inimigo. Se ele estiver parado, a direção de seu último movimento será considerada. Por isso, nunca apagamos o elemento de direção de uma unidade após uma ordem "ALTO". Finalmente, o atacante tem também um bônus se for uma unidade de cavaleiros.

A nota da unidade atacada é calculada de modo semelhante, correspondendo à soma destes elementos: um sexto da diferença de poder, a diferença entre a arma do atacante e a armadura do atacado, o moral do defensor, um bônus que depende do terreno, um bônus fixo de 2, um número aleatório de 1 a 5. Esses cálculos têm como pressuposto que defender é mais fácil que atacar. [illegible] vezes, porém, o entusiasmo do atacante pode neutralizar tal vantagem.

Assim, os atacantes obterão uma nota (AT) e os defensores, outra (DF). A unidade que alcançar o valor mais alto ganha a batalha, perdendo apenas um décimo de seu poder. O poder do derrotado é reduzido em um quinto.

Finalmente, a rotina que testa o moral é chamada para o derrotado.

O MORAL

O fator psicológico tem grande pe[illegible] [illegible] seja [illegible] seja equipado com as melhores armas do mundo, se a tropa detestar seus generais, simpatizar com a causa inimiga ou, simplesmente, não quiser lutar.

A psicologia humana é muito complexa, e não pretendemos reproduzi-la aqui. Em *Capa e Espada* o moral influencia apenas a sobrevivência da unidade e o resultado do combate.

```
[illegible]
[illegible]
[illegible]
1220 GOSUB [illegible]
1230 PRINT [illegible]
[illegible]
1240 PRINT [illegible]
ade ";un;" se desintegra"
1250 GOSUB [illegible]
1260 LET [illegible]
1270 [illegible]
[illegible]
```

```
[illegible]
[illegible]
[illegible]
1220 [illegible]
1230 LOCATE [illegible]
[illegible]
```

```
2240 PRINT "QUE A UNIDADE";UN;"
SE DESINTEGRA"
2250 GOSUB 2410
2260 T(UN,1)=5
2270 LOCATE T(UN,9),T(UN,8):PRI
NT " ":
2280 RETURN
```



```
2200  REM  MORAL
2210  IF T(UN,6) - T(UN,7) < ((
T(UN,6) / 100) * ((T(UN,5) + 2)
)) THEN  RETURN
2220  GOSUB 2540
2230  VTAB 21: PRINT "AS BAIXAS
 FORAM MUITO GRANDES"
2240  PRINT "A UNIDADE ";UN;" S
E DESINTEGROU"
2250  GOSUB 2410
2260 T(UN,1) = 5
2270 X = T(UN,9):Y = T(UN,8):N
= 14: GOSUB 10000
2280  RETURN
```



Adicione as seguintes linhas:

```
2545  COLOR= 0: FOR I = 160 TO
191
2550  HPLOT 0,I TO 279,I: NEXT
: RETURN
30000  RETURN
```



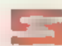
```
2200 REM  MORAL
2210 IF T(UN,6)-T(UN,7)<((T(UN,
6)/100)*((T(UN,5)+2)*10)) THEN
RETURN
2220 GOSUB 2540
2230 DRAW"BM0,152":A$="AS PERDA
S FORAM MUITO GRANDES":GOSUB 31
90
2240 DRAW"BM0,160":A$="UNIDADE"
+STR$(UN)+" SE DESINTEGRA":GOSU
B 3190
2250 GOSUB 2410
2260 T(UN,1)=5
2270 X9=T(UN,9)*8:Y9=T(UN,8)*8:
LINE(X9,Y9)-(X9+7,Y9+7),PRESET,
■
2280 RETURN
```

A linha 2210 subtrai o poder atual do poder inicial da tropa, comparando o resultado com o moral. Se a unidade sofreu uma perda de 30% e tem o moral baixo, ela se dispersa e não participa mais do jogo. Se o moral for mais elevado, uma unidade poderá perder até 70% do poder antes de se dispersar.

Quando uma unidade não passa no teste do moral, as linhas 2230 e 2240 transmitem uma mensagem informando a dispersão. O elemento de comando da matriz da tropa passa a valer 5, o que significa que a tropa bateu em retirada. A unidade é apagada da tela e ignorada. Porém, como há desertores espalhados pelo campo de batalha, a unidade poderá dificultar o movimento de tropas, apesar de estar invisível.

ENFIM, A PAZ

Entre os últimos detalhes que precisamos incluir no programa está uma rotina de verificação do fim do jogo.



```
2290 REM  Vitoria
2300 LET gd=0: LET bd=0
2310 FOR m=1 TO 8
2320 IF T(m,1)<>5 THEN  LET gd=
gd+1
2330 IF T(m+8,1)<>5 THEN  LET b
d=bd+1
2340 NEXT m
2350 IF gd>bd*2 OR (bd<2 AND gd
>2) THEN  LET vc=1
2360 IF bd>gd*2 OR (gd<2 AND bd
>2) THEN  LET de=1
2370 RETURN
2380 REM  Fim
2390 IF vc=1 THEN  PRINT "VITOR
IA !!!"
2395 IF de=1 THEN  PRINT "Uma h
umilhante derrota."
2400 RETURN
```



```
2290 REM VITORIA
2300 GD=0:BD=0
2310 FOR M=1 TO 8
2320 IF T(M,1)<>5 THEN GD=GD+1
2330 IF T(M+8,1)<>5 THEN BD=BD+
1
2340 NEXT M
2350 IF GD>BD*2 OR (BD<2 AND GD
>2) THEN VC=1
2360 IF BD>GD*2 OR (GD<2 AND BD
>2) THEN DE=1
2370 RETURN
2380 REM FIM
2390 IF VC=1 THEN PRINT "VITORI
A !!!"
2395 IF DE=1 THEN PRINT "UMA HU
MILHANTE DERROTA."
2400 RETURN
```



```
2290  REM  VITORIA
2300 GD = 0:BD = 0
2310  FOR M = 1 TO 8
2320  IF T(M,1) <  > 5 THEN GD
= GD + 1
2330  IF T(M + 8,1) <  > 5 THEN
 BD = BD + 1
2340  NEXT M
2350  IF GD > BD * 2 OR (BD < 2
 AND GD > 2) THEN VC = 1
2360  IF BD > GD * 2 OR (GD < 2
 AND BD > 2) THEN DE = 1
2370  RETURN
2380  REM  FIM
2390  IF VC = 1 THEN  PRINT "VI
TORIA !!!": GOSUB 30000
2395  IF DE = 1 THEN  PRINT "UM
A HUMILHANTE DERROTA": GOSUB 30
000
2400  RETURN
```



Modificações do programa do Apple:

```
2390  IF VC = 1 THEN  PRINT "VI
TORIA !!!"
2395  IF DE = 1 THEN  PRINT "UM
A HUMILHANTE DERROTA"
```



```
2290 REM  VITORIA
2300 GD=0:BD=0
2310 FOR M=1 TO 8
2320 IF T(M,1)<>5 THEN GD=GD+1
2330 IF T(M+8,1)<>5 THEN BD=BD+
1
2340 NEXT M
2350 IF GD>BD*2 OR (BD<2 AND GD
>2) THEN VC=1
2360 IF BD>GD*2 OR (GD<2 AND BD
>2) THEN DE=1
2370 RETURN
2380 REM  FIM
2390 IF VC=1 THEN DRAW"BM0,176"
:A$="VITORIA !!!":GOSUB 3190
2395 IF DE=1 THEN DRAW"BM0,176"
:A$="UMA HUMILHANTE DERROTA.":G
OSUB 3190
2400 RETURN
```

A rotina verifica se um dos adversários conta com o dobro de unidades do outro ou se um deles tem menos de duas unidades. Uma mensagem informa aos jogadores o resultado.

Poderíamos acrescentar outras condições de vitória. Por exemplo, um fator que decidiu muitas guerras do período medieval foi a morte do líder. Porém, a incorporação desse elemento a um jogo, além de trazer outros problemas, faria com que a atenção dos jogadores estivesse permanentemente voltada para uma das unidades.

Uma unidade também poderia obter a vitória se atingisse a extremidade oposta do mapa com metade de seus efetivos, ou metade de seu poder. Outro critério seria pelo total de baixas. Nesse caso, teríamos de acrescentar uma variável para calcular o número.

ALEA JACTA EST

Já digitamos todas as rotinas necessárias ao jogo. Agora, precisamos apenas incluir um laço principal, que as chame ordenadamente:



```
10 CLEAR
30 GOSUB 190
```

```
 40 GOSUB 470
 50 GOSUB 860
 60 REM  Loop
 70 FOR i=1 TO 8
 80 IF T(i,1)<4 THEN  GOSUB
 1380: IF y>2 THEN  GOSUB 1900
 90 IF T(i,1)<5 THEN  INK 1:
 PRINT AT T(i,8),T(i,9);u$(i)
100 NEXT i
110 FOR e=9 TO 16
120 IF T(e,1)<4 THEN  GOSUB
 2140
130 NEXT e
140 GOSUB 1020
150 GOSUB 2290
160 IF vc<>1 AND de<>1 THEN
GOTO 60
170 GOSUB 2380
180 STOP
2410 REM  Atraso
2420 PRINT AT 21,7;"[QUALQUER T
ECLA]"
2425 LET g$=INKEY$: IF g$="" TH
EN  GOTO 2425
2430 RETURN
```



```
10 CLEAR 5000
30 GOSUB 190
40 GOSUB 470
50 GOSUB 860
70 FOR I=1 TO 8
80 IF T(I,1)<4 THEN GOSUB 1380:
IF YW>2 THEN GOSUB 1900
90 IF T(I,1)<5 THEN LOCATE T(I,
9),T(I,8):PRINT CHR$(U(I));
100 NEXT I
110 FOR E=9 TO 16
120 IF T(E,1)<4 THEN GOSUB 2140
130 NEXT E
140 GOSUB 1020
150 GOSUB 2290
160 IF VC<>1 AND DE<>1 THEN 60
170 GOSUB 2380
180 END
2410 REM ATRASO
2420 LOCATE 5,22:PRINT "APERTE
QUALQUER TECLA"
2425 G$=INKEY$:IF G$="" THEN 24
25
2430 RETURN
```



```
30  GOSUB 190
40  GOSUB 470
50  GOSUB 860
70  FOR I = 1 TO 8
80  IF T(I,1) < 4 THEN  GOSUB 1
380: IF YW > 2 THEN  GOSUB 1900
90  IF T(I,1) < 5 THEN X = T(I,
9):Y = T(I,8):N =  VAL ( MID$ (
U$,I,1)): GOSUB 10000
100  NEXT I
110  FOR E = 9 TO 16
120  IF T(E,1) < 4 THEN  GOSUB
2140
130  NEXT E
140  GOSUB 1020
150  GOSUB 2290
160  IF VC <  > 1 AND DE <  > 1
 THEN  GOTO 70
170  GOSUB 2380
180  END
475  REM  XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
476  REM  XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
2410  REM   ATRASO
2420  PRINT "APERTE QUALQUER TE
CLA";: GOSUB 30000
2425  GET G$:
2430  RETURN
30000  FOR QQ = 1616 TO 2000 ST
EP 128
30010  FOR WW = 0 TO 39
30020  POKE QQ + WW + 1024, PEE
K (QQ + WW)
30030  NEXT WW,QQ
30040  RETURN
```



Modificações a serem feitas no programa acima:

```
2420  PRINT "APERTE QUALQUER TE
CLA";
```



```
10 CLEAR 500:PMODE 3,1:COLOR2,1
:PCLS:SCREEN 1,0:DU=RND(-TIMER)
18 GOSUB 2410:CLS:END
30 GOSUB 190
40 GOSUB 470:GOSUB 3130
50 GOSUB 860
60 REM
70 FOR I=1 TO 8
80 IF T(I,1)<4 THEN GOSUB 1380:
IF Y>2 THEN GOSUB 1900
90 IF T(I,1)<5 THEN COLOR 3:DRA
W"BM"+STR$(T(I,9)*8)+","+STR$(T
(I,8)*8):UU=VAL(MID$(U$,I,1)):A
$=UC$(UU):GOSUB 3000
100 NEXT I
110 FOR E=9 TO 16
120 IF T(E,1)<4 THEN GOSUB 2140
130 NEXT E
140 GOSUB 1020
150 GOSUB 2290
160 IF VC<>1 AND DE<>1 THEN 60
170 GOSUB 2380
190 REM  INICIO
200 VC=0:DE=0
2410 REM  ATRASO
2420 DRAW"BM80,176":A$="APERTE
QUALQUER TECLA":GOSUB 3190
2425 G$=INKEY$:IF G$="" THEN 24
25
2426 LINE(0,176)-(255,183),PRES
ET,BF
2430 RETURN
```

Em primeiro lugar, todos os microcomputadores, com exceção dos pertencentes às linhas Apple e TK-2000, limpam a memória. O TRS-Color seleciona o modo gráfico. Em seguida, as rotinas que criam os blocos gráficos e desenham o mapa são chamadas.

O laço principal começa na linha 60 e termina na linha 160. Ele permite que tanto o jogador quanto o computador dêem ordens às suas respectivas unidades. Depois, a rotina que faz com que as unidades obedeçam às ordens é executada dezesseis vezes, seguida de uma chamada para verificação de final de jogo. O laço se repete enquanto um dos lados não vencer.

Ao final da listagem (começando na linha 2410), temos uma rotina encarregada de provocar uma pausa em determinados momentos do jogo.

AS INSTRUÇÕES

Ao ser executado, o programa desenha o mapa com os diferentes terrenos, a moldura e as tropas beligerantes.

Este é o momento de planejar sua estratégia. Uma série de mensagens surgirá na janela de texto. Começando com a unidade um, são mostrados na tela o número e o tipo de homens da unidade, juntamente com suas últimas ordens — "ALTO", por exemplo. O jogador é então questionado sobre uma possível mudança de ordens.

Se a resposta for Sim, um menu com as opções FOGO, ALTO, MARCHE e STATUS é exibido. A opção de atirar aplica-se somente aos arqueiros e, se selecionada indevidamente para qualquer outra unidade, surgirá na tela a mensagem "SEM ARCOS". Quando a opção MARCHE for selecionada, o computador pedirá que se indique a direção (N,S,L,O).

Esse processo se repete para cada unidade, cuja cor é modificada para orientar o jogador.

O QUE FALTA?

Neste estágio, *Capa e Espada* é um jogo de guerra bem simples, mas, ainda assim, bastante divertido.

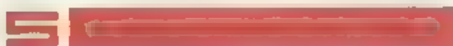
Na última parte da série, veremos como transformar o computador em um jogador mais habilidoso. Mas, especialmente se você for um principiante, tente jogar *Capa e Espada* como está, para apreciar os principais aspectos desse tipo de jogo.

AVALANCHE: O TEMPO FECHA

Meteorologia pode não ser seu ponto forte, mas chegou a hora de adicionar um pouco de tempestade ao jogo. Avalanche tem agora nuvens ou sol, conforme a versão do programa.

Como se Willie já não tivesse problemas suficientes com os buracos, as serpentes e a maré, uma nuvem escura promete chuva, ameaçando molhar seu lanche! Mas não se preocupe se você não for usuário do Spectrum ou do MSX. Troveja apenas nas versões de *Avalanche* para esses micros.

Willie está com mais sorte no programa do TRS-Color. Não há nuvens no céu e um belo sol de verão brilhará durante todo o jogo.

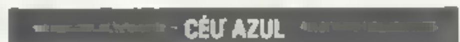


A seguinte rotina desenha uma nuvem que se movimenta de acordo com a direção do vento:

```
 10 REM org 58795
 20 REM cld ld a,(57347)
 30 REM dec a
 40 REM ld (57347),a
 50 REM cp 0
 60 REM jr z,cdm
 70 REM ret
 80 REM cdm ld a,6
 90 REM ld (57347),a
100 REM ld a,45
110 REM ld bc,16384
120 REM ld hl,(57345)
130 REM ld d,3
140 REM ld e,2
150 REM call blk
160 REM ld a,(57348)
170 REM cp 0
180 REM jr z,crt
190 REM dec hl
200 REM jr chm
210 REM crt inc hl
220 REM chm ld (57345),hl
230 REM ld bc,57144
240 REM ld a,47
250 [redacted] ld d,3
260 REM ld e,2
270 REM call blk
280 REM ld de,129
290 REM sbc hl,de
300 REM jr nz,cnr
310 REM ld a,0
320 REM ld (57348),a
330 REM ret
340 REM cnr ld de,144
350 REM ld hl,(57345)
360 REM sbc hl,de
370 REM jr nz,cnl
380 REM ld a,1
390 REM ld (57348),a
400 REM cnl ret
410 REM org 58970
420 REM blk *
```

A posição de memória 57347 contém o atraso da nuvem. Essa variável controla a rapidez com que ela se movimenta pelo céu. Quando se inicializa o nível do jogo, a velocidade é especificada por meio de um número que é carregado nessa posição.

Depois de carregado no acumulador, o atraso da nuvem é decrementado e armazenado de volta em 57347. Se seu valor tiver sido reduzido a zero, as instruções **cp 0** e **jr z** saltam a instrução **ret** e dão prosseguimento à rotina que movimenta a nuvem. Caso contrário, o processador retorna e adia o movimento da nuvem até que o valor do atraso tenha sido reduzido a zero.



CÉU AZUL

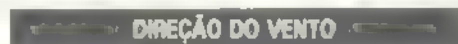
A rotina encarregada de movimentar a nuvem começa acertando a variável de atraso. Repare que, se o valor desta já foi reduzido a 0, um outro decremento resultaria em 255. Para que o atraso chegasse novamente a 0, a rotina precisaria ser chamada 256 vezes — o que tornaria o movimento da nuvem extremamente lento. Para evitar que isso ocorra, o número 6 é carregado no acumulador e colocado no endereço 57347. Com o atraso assim restabelecido, uma suave e refrescante brisa passa a empurrar a nuvem, o que você pode constatar observando a tela.

O número 45 é carregado no acumulador. Ele será aproveitado pela rotina **blk**, que imprime um bloco de caracteres. As dimensões do bloco devem ser carregadas nos registradores D e E — D carrega o número de colunas e E, o número de linhas. A rotina **blk**, por sua vez, utiliza a rotina **print**, que tem sido empregada com freqüência neste jogo. O par de registros HL deve, então, carregar a posição de tela; o par BC, o apontador de dados, e o acumulador A, a cor do caractere.

O número 5 — código da cor azul ciano sobre azul ciano — é carregado no acumulador, pois a rotina imprime caracteres de céu sobre a nuvem, apagando-a. Depois disso, imprime a nuvem uma posição para a direita ou para a esquerda, dependendo da direção em que o vento está soprando.

O par de registros BC é carregado com o endereço inicial dos padrões. Esse endereço irá fornecer os dados apropriados ao bloco azul de céu. O par HL é carregado com o conteúdo de 57345 e 57346, que são os apontadores da posição atual da nuvem.

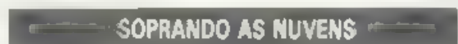
A nuvem tem três colunas de comprimento por duas linhas de altura. Assim, 3 é carregado em D e 2, em E. Em seguida, a rotina **blk** é chamada e imprime o bloco de caracteres de céu azul, apagando a nuvem.



DIREÇÃO DO VENTO

A nuvem deve se movimentar na direção em que o vento está soprando. Para isso, utiliza-se a posição de memória 57348, originalmente ajustada pela rotina de inicialização, como uma baliza. Se ela contém o valor 1, o vento está soprando para a esquerda; se contém o valor 0, o vento está soprando para a direita.

O conteúdo de 57348 é carregado no acumulador. Se seu valor for 0, a instrução **jr z** salta para o rótulo **crt** e o apontador de posição da nuvem é incrementado, movendo-se uma posição para a direita na tela. Se o conteúdo de 57348 for 1, a instrução **jr z** não atua e o par de registros HL é decrementado, movendo o apontador de posição da nuvem um caractere para a esquerda. A próxima instrução **jr z** simplesmente salta sobre a instrução **inc**.



SOPRANDO AS NUVENS

O conteúdo de HL é trazido de volta para o apontador de posição da nuvem, nos endereços 57345, ajustando-o. Em seguida, o apontador de dados BC é carregado com o endereço inicial dos dados da nuvem.

O acumulador é carregado com 47 — código de branco sobre fundo ciano, a cor da nuvem. O registro D é carregado com 3 e o registro E, novamente com 2.

- DESENHO DA NUVEM
- DIREÇÃO DO VENTO
- OS CARACTERES DO CÉU AZUL
- O EFEITO DO ATRASO
- MUDANÇA DE DIREÇÃO
- CÉU ENCOBERTO
- O MOVIMENTO DAS NUVENS
- VERIFICAÇÃO DA POSIÇÃO
- O SOL NO TRS-COLOR

A nuvem é do mesmo tamanho daquela que foi apagada.

Depois disso, a rotina **blk** é chamada e imprime a nuvem em sua nova posição na tela.

O movimento da nuvem pode levá-la até o canto do vídeo. A parte dela que ultrapassar a primeira ou a última coluna será impressa na linha seguinte ou na linha de cima, provocando a desfiguração de sua forma.

Para evitar que isso aconteça, o programa verifica primeiro se a nuvem alcançou o extremo esquerdo do vídeo. O par DE é carregado com 129 — a posição de tela do canto esquerdo da linha. Esse valor é subtraído do apontador de posição da nuvem no par HL. Se o resultado não for 0, a nuvem não chegou no canto esquerdo. A instrução **jr nz** salta então para verificar se ela alcançou o canto direito.

Se o resultado for 0, a nuvem atingiu o canto esquerdo e o salto não ocorre. O acumulador é então carregado com 0 e esse valor é transferido para a baliza de direção do vento, de modo que a nuvem se desloque para a direita na próxima vez.

A rotina **crr** verifica se a nuvem chegou ao canto esquerdo do vídeo subtraindo o número 144 do apontador de posição. Se ela atingiu o canto direito da linha, o valor 1 é carregado no indicador da direção do vento. Na próxima vez, a nuvem irá para a esquerda.



O BLOCO DA NUVEM

Esta rotina imprime um bloco de caracteres na tela, com D colunas de comprimento por E linhas de altura.

```
 10 REM org 58970
 20 REM blk push hl
 30 REM blj push de
 40 REM push hl
 50 REM z push de
 60 REM call print
 70 REM inc hl
 80 REM pop de
 90 REM dec d
100 REM jr nz,z
110 REM pop hl
120 REM ld de,32
130 REM add hl,de
140 REM pop de
150 REM dec e
160 REM jr nz,blj
170 REM pop hl
180 REM ret
190 REM org 58217
200 REM print *
```

O programa anterior, que movimenta a nuvem, utiliza a rotina **blk**. Portanto, não irá trabalhar até que a rotina que acabamos de apresentar esteja guardada na memória. Esta, por sua vez, não entrará em funcionamento sem que a rotina **print**, que também é chamada, se encontre na memória.

É O BLOCO CERTO?

O apontador da posição de tela em HL e o número de colunas e linhas que estão em DE são armazenados na pilha duas vezes. Isso é feito porque a rotina trabalhará com duas dimensões, usando dois laços.

Como os parâmetros já estão armazenados, a rotina **print** é chamada e imprime o primeiro caractere na tela. O par HL é incrementado, movendo-se um caractere para a direita. As dimensões do bloco são recuperadas da pilha e o parâmetro horizontal — equivalente ao número de colunas — é decrementado. Se o parâmetro não foi reduzido a 0, a instrução **jr nz** volta ao início do laço para imprimir outro caractere da nuvem nessa linha da tela.

Quando o registrador D for 0, a primeira linha do bloco estará completa. Em seguida, recupera-se o par HL da pilha e adiciona-se o número 32 ao seu valor. Como se trata de uma adição de números de dezesseis bits, a operação é feita por intermédio do par DE. O resultado fica no par HL, fazendo com que esse apontador de tela se mova uma linha para baixo. O par DE é novamente recuperado da pilha, para ajustar o valor em D e para decrementar o valor em E. Esse registrador conta o número de linhas que falta.

Se o valor no registro E não tiver sido reduzido a 0, a instrução **jr nz** manda o processador de volta ao início do laço, para começar a impressão de uma nova linha.

Quando o valor de E for 0, o processador sai do laço e o par HL é recuperado da pilha, ajustando o apontador de tela para as checagens que serão realizadas. Lembre-se de que precisamos verificar se a nuvem não atingiu nenhum dos cantos da tela.

Depois disso, a rotina **blk** retorna ao programa principal deste artigo.



Na versão de *Avalanche* para o TRS-Color, a inclusão de uma nuvem exigiria um conjunto adicional de dados ou padrões. Ainda que isso não constituísse um problema, haveria um outro obstáculo: o conjunto de cores já definido é azul, vermelho e verde — nenhuma delas, portanto, é adequada para a nuvem. Como nesta versão a aventura transcorre num dia muito quente — com o céu avermelhado —, uma nuvem não é realmente essencial. A rotina a seguir movimenta o sol durante o jogo.

```
10   ORG 19727
20   MOVSUN DEC 18258
30   BNE SUNRET
40   LDA #5
50   STA 18258
60   SYNC
70   LDX #1569
80   LDA #30
90   MOVA PSHS A
100   LDA #2
110   MOVB ANDCC #$FE
120   PSHS CC
130   CLRB
140   MOVC PULS CC
150   ROR B,X
160   PSHS CC
170   INCB
180   CMPB #14
190   BNE MOVC
200   LSL,X
210   PULS CC
220   ROR,X
230   DECA
240   BNE MOVB
250   LEAX 32,X
260   PULS A
270   DEC A
280   BNE MOVA
290   SUNRET RTS
```

Para testar a rotina, digite o seguinte programa:

```
10 EXEC 19426
20 EXEC 19727
30 GOTO 20
```

A posição de memória 18258 contém a variável de atraso do sol. Essa variável impede que o atraso risque o céu como se fosse um disco voador, e é ajustada, inicialmente, com 5.

A primeira instrução da rotina decrementa o atraso do sol. Se o valor dessa variável for 0, a instrução **BNE** é desviada para a instrução **RTS** do final da rotina e o processador retorna. Caso contrário, a instrução **BNE** não realiza o desvio e o processador continua com o restante da rotina. Em outras palavras, o movimento do sol sofre um atraso, já que a rotina é executada uma vez a cada cinco chamadas.

Logo que é chamada, a rotina ajusta o atraso do sol, colocando o número 5 no acumulador e armazenando-o de volta na posição de memória 18258.

SINCRONISMO

O comando **SYNC** encarrega-se de sincronizar o restante da rotina com o sinal de TV. Isso é feito para resguardar a posição do sol da mudança que se efetuará na tela. O registro X é carregado com o endereço na tela do canto superior esquerdo do sol. O acumulador

A é carregado com 30 (o sol tem 30 linhas de altura), valor que se guarda na pilha. Em seguida, A é carregado com 2 — um contador de laço que fará o sol se deslocar lateralmente dois pontos na tela.

A operação AND é efetuada entre o registro de código condicional e o número hexadecimal $FE. Como a representação binária de $FE é 11111110, os sete bits mais significativos do registro de código condicional permanecem inalterados, enquanto o bit menos significativo é ajustado com 0. Esse bit — ou seja, o bit 0 do registro de código condicional — corresponde à baliza **carry**, que, portanto, é zerada nessa operação.

O registro de código condicional é guardado na pilha. O registrador B, ajustado com 0, será usado como um compensador. A rotina está ajustada e pronta para realizar o deslocamento.

ROTAÇÃO DO SOL

A parte seguinte da rotina que estamos examinando desloca o sol pelo céu. Para isso recupera da pilha o registro de código condicional, garantindo que, na primeira passagem do laço, o valor da baliza **carry** seja zero.

A instrução **ROR B,X** roda uma posição para a direita os bits do endereço da tela apontado por X+B. A rotação desloca todos os bits. Assim, o último lugar desocupado no processo é carregado com o conteúdo da baliza **carry**, e o bit que ultrapassou o fim do outro lado é colocado nessa baliza.

Os pontos que formam essa parte do sol são deslocados uma posição para a direita. O ponto no canto esquerdo é aceso (ajustado para 1) ou apagado (ajustado para 0, ou seja, adquire a cor do céu). O resto desse byte do sol é deslocado um ponto.

O ponto colocado na baliza **carry** é preservado e a baliza volta para a pilha. O contador de laço em B é incrementado e comparado a 14 — número de caracteres da parte do céu ocupada pelo sol. A comparação permite-nos verificar se o último byte da área já foi deslocado. Se B for menor que 14, a instrução **BNE** manda o processador de volta, para continuar com o próximo caractere. Chegando ao último caractere, o processador salta do laço.

FINAL LIVRE

Esta aventura se passa no hemisfério norte e o sol se move da esquerda para a direita. Seria muito artificial se, ao alcançar o final da tela ou esbarrar no escore, o sol começasse a fazer o caminho de volta — isto é, da direita para a esquerda.

Porém, como não existe noite na região de *Avalanche*, nosso astro reaparecerá à esquerda sempre que concluir seu percurso na tela.

A instrução **LSL ,X** promove o deslocamento lógico para a esquerda do conteúdo da posição de tela apontada por X. O registrador X não foi alterado nesta parte da rotina, e, por isso, ainda aponta para o canto inferior esquerdo da figura do sol. Assim, o bit do canto esquerdo é deslocado para fora do registrador. O bit do canto direito, por sua vez, que já tinha sido deslocado do byte, é recuperado da pilha, voltando para a baliza **carry**. Em seguida, a instrução **ROR ,X** faz com que ele execute uma rotação sobre o canto esquerdo da posição de tela.

O contador em A é decrementado e, se seu valor ainda não tiver chegado a 0, o processador salta para deslocar o sol mais um ponto.

Quando o deslocamento corresponder a dois pontos, a instrução **LEAX 32,X** adiciona o valor 32 ao conteúdo do registro X e movimenta o apontador da posição de tela para a próxima linha abaixo do sol. O contador de linha, recuperado da pilha para o acumulador, é, então, decrementado. Se seu conteúdo não for igual a 0, a instrução **BNE** manda o processador de volta para rolar a próxima coluna de caracteres. Caso contrário, o processador salta para a instrução **RTS** e retorna.

A rotina a seguir coloca na tela uma nuvem que se move pelo céu, empurrada pelo vento:

```
10    org 54270
20    ld a,(-5208)
30    dec a
40    ld (-5208),a
50    cp 0
60    jr z,cl
70    ret
80    cl ld a,6
90    ld (-5208),a
100    ld a,(-5207)
110    cp 0
120    jr z,ct
130    ld hl,-5210
140    dec (hl)
150    jr cm
160    ct ld hl,-5210
170    inc (hl)
180    cm ld hl,(62413)
190    ld b,6
200    ld a,(-5210)
210    ld c,a
220    ld d,20
230    ld e,1
240    call  11168
250    ld hl,(62413)
260    ld de,4
270    add hl,de
280    ld b,6
290    ld a,(-5210)
300    add a,16
310    ld c,a
320    ld d,24
330    ld e,1
340    call  11168
350    ld a,(-5210)
360    cp 2
370    jr nz,cr
380    ld a,0
390    ld (-5207),a
400    ret
410    cr ld a,(-5210)
420    cp 230
430    jr nz,cf
440    ld a,1
450    ld (-5207),a
460    cf ret
470    end
```

O endereço −5208 contém a variável de atraso da nuvem. Essa variável controla a rapidez com que a nuvem se move. A velocidade foi especificada na parte da rotina principal que inicializa as variáveis, com a armazenagem do número 6 naquele endereço.

O atraso da nuvem é carregado no acumulador, decrementado e novamente armazenado em −5208. Quando a variável é reduzida a 0, as instruções **cp 0** e **jr z** saltam a instrução **ret**, e o processador executa o restante da rotina que movimenta a nuvem. Enquanto isso não ocorre, o processador retorna e adia o movimento da nuvem. Em outras palavras, a rotina é executada uma vez a cada seis chamadas. Utilizamos essa técnica de controle do sincronismo em várias partes deste jogo.

DIREÇÃO DO VENTO

Antes de mais nada, a rotina acerta o atraso da nuvem. Lembre-se de que, para que ela prossiga, o conteúdo da variável de atraso precisa ser reduzido a zero. Um outro decremento teria como resultado 255, pois, na representação binária adotada pelo processador, −1 é o mesmo que 255. Assim, para uma nova execução, a rotina teria que ser chamada 255 vezes, o que resultaria num movimento extremamente lento da nuvem. Por esse motivo, o número 6 é carregado em −5208 pelo acumulador.

Em seguida, o conteúdo de −5207 é colocado no acumulador. Essa variável contém a direção do vento. Se for 0, o

vento sopra para a direita; se for 1, para a esquerda. O endereço −5210 contém a posição horizontal da nuvem. Apenas essa posição precisa ser alterada, já que a nuvem se move na mesma linha, de um lado para outro.

O valor de −5207 é testado pela instrução **cp 0**. Se for 0, a instrução **jr z** salta para o rótulo **ct**, o valor em −5210 é incrementado através de HL e a nuvem se move para a direita. Se for 1, o salto não ocorre, o valor em −5210 é decrementado através do par HL e a nuvem se move para a esquerda.

FORMAÇÃO DA NUVEM

Utilizamos sprites para imprimir a nuvem na tela do MSX. No final deste artigo, você terá explicações mais detalhadas sobre o emprego desse recurso gráfico em código de máquina.

Para que esta parte do programa funcione, os padrões das figuras devem estar na memória. Além disso, é preciso que a rotina −12121 — que transfere esses padrões da RAM para a tabela de padrões de sprites na VRAM — tenha sido executada. Se você estiver acompanhando a seqüência do jogo e executar a rotina principal antes da que apresentamos neste artigo, as condições acima serão cumpridas.

A rotina −11168 é utilizada duas vezes para colocar na tela os dois sprites que compõem a nuvem. Assim, os parâmetros necessários precisam ser carregados nos registros adequados.

O par HL deve conter o endereço inicial do sprite na Tabela de Atributos de Sprites (TAS). O endereço inicial dessa tabela está armazenado nas posições 62413 e 62414. Como a nuvem é a primeira figura do jogo que utiliza o sprite, este será o endereço do primeiro sprite da nuvem. O registro B, que deve conter a coordenada Y do sprite na tela, é ajustado com o valor 6. O registro C conterá a coordenada X. Para isso, o conteúdo de −5210 é colocado em C pelo acumulador. O registro D contém o código do padrão de 32 bytes que forma o sprite — no nosso caso, a primeira parte da nuvem tem o código 20. Finalmente, o registro E deve conter a cor do sprite — usamos aqui o preto, cujo código é 1.

Para o sprite que completa nuvem há algumas alterações. Como se trata do segundo sprite do jogo, soma-se 4 ao endereço inicial da TAS no par HL. A coordenada Y em B permanece 6. A coordenada X em C é o resultado da soma do conteúdo anterior desse registro com 16. A operação é necessária porque imprimimos o segundo sprite que compõe a nuvem ao lado direito do primeiro, que tem dezesseis bits de comprimento. O código do segundo sprite da nuvem é 24, valor colocado em D. Sua cor é a mesma — ou seja, o registro E continua contendo o valor 1.

SOPRANDO O VENTO

É importante checar se a nuvem chegou a um dos cantos do vídeo. Caso isso ocorra, estaremos decrementando o valor 0 ou incrementando o valor 255 na coordenada horizontal, o que resultará em erro nessa coordenada.

Verificamos primeiro se o conteúdo de −5210 é 2 ou 230. No primeiro caso, alteramos a direção do vento em −5207 para 0, ou seja, para a direita; no segundo, alteramos para 1, ou seja, para a esquerda. Se o endereço −5210 não contiver nenhum desses dois valores, a direção não é alterada e o processador retorna.

UTILIZAÇÃO DE SPRITES

O sprite é um bloco de 16 X 16 pontos ou de 8 X 8 pontos, dependendo do tamanho selecionado. No primeiro caso, seus padrões ocupam 32 bytes, o que nos permite definir apenas 64 sprites diferentes. No segundo caso, os padrões ocupam oito bytes, como um caractere gráfico comum, e podemos definir até 256 sprites diferentes. Esses padrões devem ser colocados na tabela de padrões de sprites, cujo endereço inicial está armazenado nas posições 62415 e 62416 da RAM.

Os sprites podem ser quatro vezes maiores do que os blocos gráficos comuns, mas não é só isso o que os distingue. Normalmente, para movimentar um caractere gráfico na tela, temos que apagá-lo de sua posição anterior e imprimi-lo na nova posição. Um sprite pode ser movimentado facilmente pela tela: alterando-se suas coordenadas X e Y de impressão, ele será automaticamente apagado da posição que ocupava e impresso na nova posição. Tudo isso é feito pelo processador de vídeo.

Esse recurso gráfico tem, entretanto, suas limitações. Os sprites são definidos numa tabela de 256 bytes, a já mencionada Tabela de Atributos de Sprites — TAS. Os atributos de cada sprite ocupam quatro bytes; portanto, não podemos colocar ao mesmo tempo na tela mais do que 32 sprites diferentes, independentemente do tamanho que tenha sido selecionado.

O primeiro byte guarda a coordenada Y; o segundo, a coordenada X; o terceiro, o número do sprite (é possível escolher entre 64 e 256 padrões diferentes, dependendo do tamanho escolhido); o quarto e último, a cor. Como o sprite tem apenas cor de frente, os bits apagados são transparentes na tela, o que nos permite criar interessantes efeitos visuais.

Uma dúvida pode surgir: o que acontece se as coordenadas de dois ou mais sprites coincidirem? Os sprites são hierarquizados segundo a ordem em que foram definidos na TAS — ou seja, o sprite que ocupa os quatro primeiros bytes na TAS tem precedência sobre o que ocupa os quatro bytes seguintes e assim por diante. Em resumo, o sprite que tem precedência aparecerá na frente dos demais. A justaposição de figuras assim obtida muitas vezes é aproveitada na composição de efeitos visuais. Mas não se esqueça de que só podemos ter até quatro sprites numa mesma linha. Se tentarmos colocar um quinto sprite, o último na ordem hierárquica desaparecerá.

A rotina a seguir coloca na TAS os atributos de um sprite, ou seja, coloca um sprite na tela.

```
org -11168
ld a,b
push de
push bc
push hl
call 77
pop hl
pop bc
inc hl
 ld a,c
 push hl
 call 77
 pop hl
 pop de
 inc hl
 ld a,d
 push de
 push hl
 call 77
 pop hl
 pop de
 inc hl
 ld a,e
 call 77
 ret
 end
```

Essa rotina utiliza os parâmetros fornecidos pelo par HL e pelos registros B, C, D e E para colocar na TAS os atributos de um sprite.

O par HL deve conter o endereço inicial do sprite na TAS; o registro B, a coordenada Y; o registro C, a coordenada X; o registro D, o código do sprite e o registro E, a cor do sprite.

AFINAL, QUAL É O SEU SOM?





O traçado imita a onda sonora original.

brações do ar em sons reconhecíveis. Nessa forma, o som é um sinal analógico — isto é, varia continuamente dentro de um intervalo, e qualquer modificação é significativa. Os computadores, ao contrário dos seres humanos, não conseguem reconhecer esse tipo de variação e precisam de um sinal digital. Assim, cada variação é representada por um valor diferente, 0 ou 1, presença ou ausência de sinal.

Mesmo que o computador não seja capaz de interpretar diretamente um sinal sonoro, não é difícil converter a onda sonora analógica de uma melodia, por exemplo, em informação digital.

Essa técnica é a última novidade nos estúdios de gravação, que estão trocando as fitas tradicionais (onde se grava o sinal analógico) por sistemas computadorizados que gravam as músicas em discos (discos de computador). A vantagem de se gravar o som na forma digital baseia-se na facilidade com que se pode combinar o sinal gravado com outros sons. Além disso, uma vez gravado nessa forma, o risco de distorção do sinal devido a limitações do equipamento é bem menor.

MÚSICA EM SEU MICRO

Embora a tecnologia necessária para esse tipo de gravação se restrinja a estúdios com equipamentos altamente sofisticados, a maioria dos microcomputadores tem os recursos básicos que permitem a exploração dessa técnica. Cada vez que carregamos um programa gravado em fita cassete, estamos reproduzindo um sinal que foi registrado na forma digital. Mas quem já ouviu uma fita de programas sabe que ela produz som, ainda que não muito agradável.

Sons e melodias podem ser colocados em seu micro, desde que você recorra às técnicas de programação adequadas (em linguagem de máquina, inclusive). Para isso, é necessário apenas introduzir um sinal analógico através do plugue de gravação e ensinar o computador a interpretá-lo, transformando-o em um sinal digital.

Uma vez feito isto, o programa poderá armazenar o sinal digital na memória, ou mesmo mostrá-lo em forma gráfica, como faz o programa deste artigo. Isso significa que o seu computador será capaz agora de transformar qualquer som em números, que podem ser usados para produzir um gráfico na tela ou ser armazenados na memória para reprodução posterior.

O TRAÇADO DO SOM

O programa pode reproduzir um traçado gráfico que corresponde ao som tocado pelo gravador. Quando este é ligado, uma série de linhas regularmente espaçadas surge na tela (quanto maior a freqüência, mais alta é a linha). No momento em que a tela fica cheia, o traçado desaparece, recomeçando do canto esquerdo do vídeo.

GRAVE O SOM

O programa também permite, numa segunda opção, a gravação digital do sinal de entrada — o tamanho do trecho que se pode gravar é bem limitado, por razões que explicaremos mais adiante. Uma terceira opção possibilita a reprodução do som obtido.

COMO FUNCIONA

Como o computador não pode interpretar diretamente o sinal sonoro analógico, nós o programamos de modo a atribuir valores digitais a esse sinal. O programa verifica repetidamente os sinais na porta de entrada do cassete em intervalos muito curtos de tempo (milhares de vezes por segundo). Esses sinais só podem ser 0 ou 1 — não há valores intermediários como no sinal analógico. A alta velocidade com que a porta é lida repetidas vezes faz com que as variações do sinal digital imitem o sinal analógico.

Imaginemos, por exemplo, que entramos com um sinal de freqüência de 256 Hz (nota C). Esse sinal atinge o pico 256 vezes por segundo, e cada pico dura 1/512 de segundo. Se fizermos a leitura da porta 2000 vezes por segundo, obteremos um pico a cada quatro leituras. É assim que a variação do sinal digital imita a onda sonora analógica — quanto mais rápidas forem as leituras, mais precisa será a conversão analógico-digital.

O programa utiliza os valores digitais obtidos da fita dos modos descritos. Para mostrar a onda sonora graficamente, ele calcula quantos picos — ou quantos "uns" — obteve por unidade de tempo, o que equivale à freqüência média daquele intervalo. Novamente, quanto mais rápida for a leitura, mais preciso será o gráfico.

O processo de gravação é semelhante. O programa toma oito leituras e as armazena em um byte de memória. Esse procedimento é repetido para cada oito leituras. Como um elevado número de leituras é feito em um curto espaço de tempo, o programa utiliza grande quantidade de memória — no Spectrum, por exemplo, toda a memória seria consumida por um trecho musical de apenas oito segundos de duração.

A reprodução do sinal digital constitui o processo inverso, no qual a informação armazenada na memória é enviada ao alto-falante para imitar as vibrações lidas na porta do cassete durante a gravação. Devido às limitações do equipamento, o som produzido está longe de ser de alta-fidelidade.

O programa do Spectrum é dividido em duas partes — um programa BASIC com as rotinas de gravação, execução e de traçado gráfico, e uma rotina em linguagem de máquina, para ler a entrada do gravador cassete.

Digite a primeira parte e grave-a utilizando a instrução:

```
SAVE 'ANALYSER' LINE 5
```

```
 10 CLEAR 26000: RESTORE · LET
 t=0: LET x=65368
 20 FOR n=1 TO 108
 30 READ a: POKE x,a
 40 LET t=t+a
 50 LET x=x+1
 60 NEXT n
 70 IF t=12721 THEN  PRINT "OK
 .": STOP
 80 PRINT "ERRO NAS LINHAS 'DA
 TA'"· STOP
 90 DATA 14,64,175,8,17,208,7,
 219,254,230,64,185,40,7,62,64
 ,169,79,8,60,8,29,32,239,175,
 178,40,5,30,255,21,24,230,8,
 203,63,6,0,79,201
100 DATA 243,33,144,101,17,80,
255,6,7,219,254,203,119,32,2,
203,254,203,62,16,244,35,125,
187,32,237,124,186,32,233,251,
201,243,33,144,101,17,80,255,6
,8,203,70,40,4
110 DATA 62,0,211,254,62,255,
211,254,203,14,16,240,35,125,
```

```
187,32,233,124,186,32,229,251,
201
```

A seguir, digite a segunda parte, que contém a rotina em código dentro de suas linhas **DATA**. A linha 80 verifica se houve algum erro de digitação nas linhas **DATA**, por meio da soma dos números. Execute (**RUN**) o segundo programa e grave a rotina em código na fita, numa posição imediatamente posterior ao primeiro programa com:

```
SAVE 'ANALYSER' CODE 65368,109
```

Ao ser rodado, o primeiro programa se auto-executará a partir da linha 5, carregando então a rotina em código. Completado o processo, surgirá na tela um menu com as três opções.

A primeira delas solicitará que conectemos o gravador e toquemos algum som. Quando fazemos isso e pressionamos qualquer tecla, é desenhado um gráfico contínuo da evolução da freqüência no tempo. Aperte M para retornar ao menu principal ou F para congelar a imagem. Poderemos descongelar a imagem apertando qualquer tecla.

Se escolhermos a opção dois, o programa solicitará que iniciemos a execução da música no gravador, informando quando a gravação estiver completa e retornando então ao menu. Podemos tocar qualquer som, mas os curtos e agudos serão melhor reproduzidos. As limitações do Spectrum fazem com que o som seja reproduzido mais lentamente do que foi gravado; além disso, os sons mais graves são filtrados — a porta de entrada do gravador não é capaz de diferenciar os sons abaixo de uma certa freqüência.

É evidente que ninguém vai querer gastar quase toda a memória do seu micro simplesmente para incorporar uns poucos segundos de um som dissonante em outros programas. Contudo, se você quiser experimentar, use:

```
SAVE 'SOM' CODE 26000,39360
```

Para gravar a rotina em código que executa o som, digite:

```
SAVE 'TOCA' CODE 65440,40
```

A memória precisa ser protegida por **CLEAR 25299**. Para ouvir o som, digite **RAND USR 65440**.

Tudo isso deixará apenas 3Kbytes de memória livres para seu programa, o que não é muito, a menos que você programe em linguagem de máquina.

```
 5 CLEAR 25999: LOAD ""CODE
10 GOSUB 200: GOSUB 500
20 IF INKEY$="" THEN  GOTO 20
21 LET C=CODE INKEY$: IF C<49
 OR C>51 THEN  GOTO 20
22 GOSUB 30: GOSUB 200
24 IF C=49 THEN  GOTO 100
25 IF C=50 THEN  GOTO 600
26 IF C=51 THEN  GOTO 700
30 FOR N=30 TO 50 STEP 3:
 SOUND .01,N: NEXT N: RETURN
100 CLS : GOSUB 1000: PRINT AT
12,4; BRIGHT 1;"PRESSIONE QUAL
QUER TECLA"
101 IF INKEY$="" THEN  GOTO
 101
102 SOUND .1,10
104 CLS : GOSUB 150: GOSUB 800
105 FOR X=0 TO 255: PLOT X,0:
DRAW 0,USR 65368
110 IF INKEY$="m" THEN  GOSUB
30: GOTO 10
111 IF INKEY$="f" THEN  GOSUB
801: GOTO 140
130 NEXT X: CLS : GOSUB 150:
GOSUB 800: GOTO 105
140 PRINT AT 0,0;"
                       ": SOUND
.1,40: PAUSE 50: IF INKEY$ ""
THEN  GOTO 140
141 SOUND .1,10: CLS : GOTO
 104
150 PRINT AT 0,2; BRIGHT 1;"
PRESSIONE (M) PARA MENU  ":
RETURN
200 BORDER 5: PAPER 5: INK 0:
CLS : RETURN
500 PRINT AT 0,2; PAPER 2; INK
7;"  MENU - ANALISADOR SONORO
 "
510 PRINT AT 5,7;"1- GRAFICO D
E BARRAS";AT 7,7;"2- GRAVAR SO
M";AT 9,7;"3- REPRODUZIR SOM"
520 PRINT AT 15,4; PAPER 4;"PR
ESSIONE (1) (2) OU (3)":
RETURN
600 PAUSE 20: GOSUB 1000:
PRINT AT 13,0;"Qualquer tecla
para GRAVAR"
605 IF INKEY$="" THEN  GOTO
 605
606 SOUND .05,20: CLS : PRINT
AT 10,8;"AGUARDE": RAND USR
65408: SOUND .1,30: CLS :
PRINT AT 10,6; BRIGHT 1;" GRAV
ACAO CONCLUIDA ": PAUSE 300:
GOTO 10
700 RAND USR 65440: GOTO 10
800 PRINT AT 1,4; PAPER 4;"  (
F) CONGELA A IMAGEM  ": RETURN

801 PRINT AT 1,0; PAPER 4;" QU
ALQUER TECLA PARA CONTINUAR ":
RETURN
1000 PRINT AT 4,2;"Conecte o te
rminal EAR do seu";AT 6,0;"grav
ador ao terminal EAR do seu";AT
 8,0;"Spectrum. E toque alguma
musica.": RETURN
```



A rotina em código de máquina que lê a entrada do gravador encontra-se a partir da linha 1000 **DATA**. A soma dos números de cada linha está em seu trecho final para verificação, a fim de evitar que um erro de digitação acabe levando a um desastre.

Digitado o programa, grave-o em disco ou fita antes de executá-lo. Quando isto é feito, um menu aparece na tela. Se houver algum erro nas linhas **DATA**, surgirá na tela uma mensagem "VERIFIQUE A LINHA (número...)".

Aperte 1 para fazer a primeira opção. O programa solicitará que posicionemos a fita e liguemos o gravador. Ao pressionarmos a tecla **<ENTER>**, a leitura da fita começará, juntamente com o desenho do gráfico correspondente. Aperte M para retornar ao menu ou **<SHIFT><@>** para congelar a imagem.

Digite 2 para fazer a segunda opção (gravar um trecho). O programa faz as mesmas solicitações da opção 1. Enquanto a música estiver sendo executada, um gráfico colorido aparecerá na tela. Quando a gravação estiver completa, o gráfico será apagado e o programa retornará ao menu.

Aperte 3 para reproduzir a música

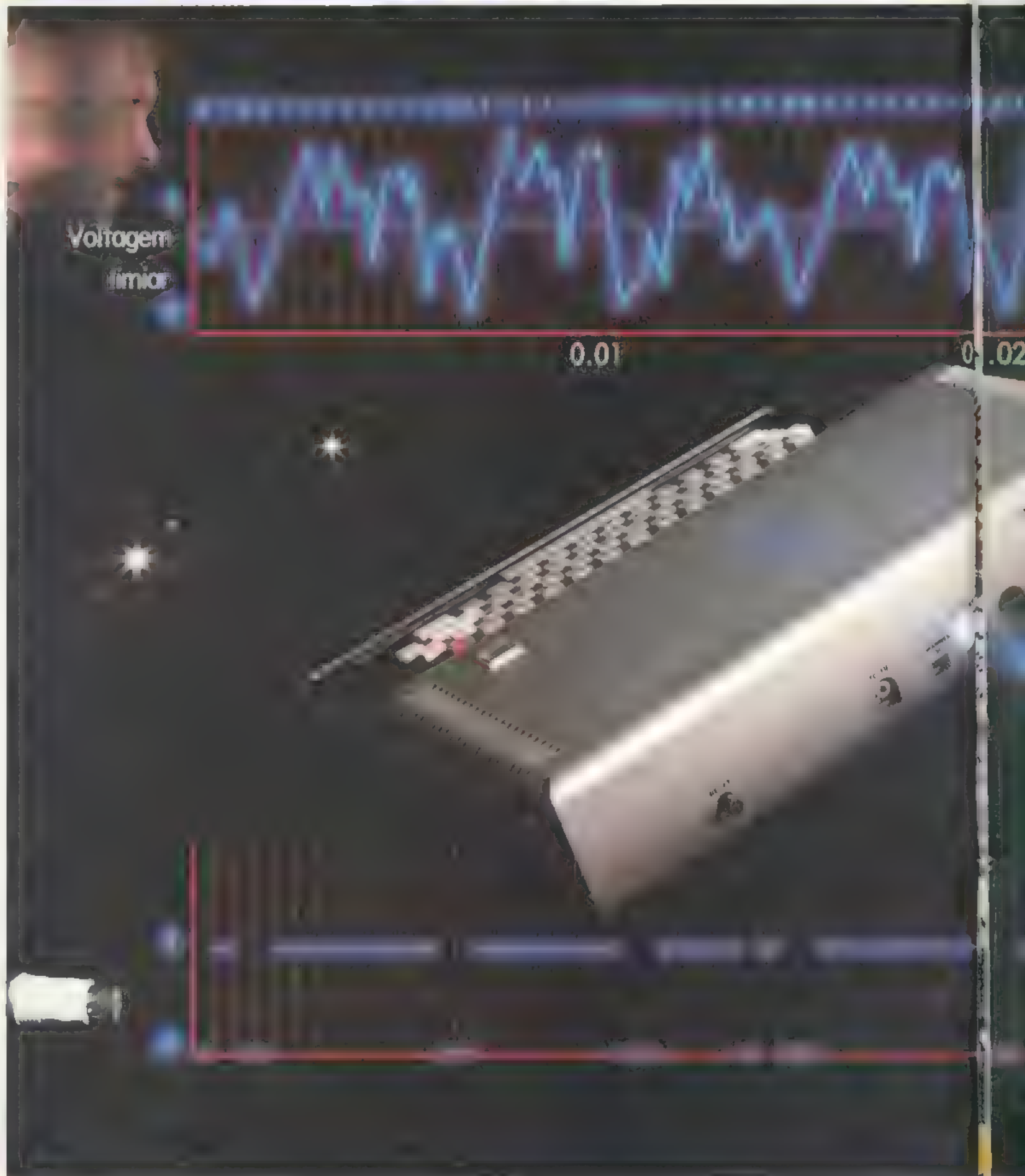


O que é análise espectral?

A análise espectral consiste na obtenção dos componentes de frequência pura, presentes em uma determinada onda sonora. Explicando melhor: uma onda sonora complexa, como o som de uma flauta, a voz humana, o ruído de uma britadeira etc. são misturas de diversas freqüências sonoras puras. Por exemplo, em uma certa fração de tempo (medida em milissegundos), pode haver 12% de frequência 1000 Hertz (ciclos por segundo de uma onda senoidal), 8% de 1100 Hertz etc.

O objetivo da análise espectral é quantificar cada freqüência sonora pura presente em um som. Essa quantificação, feita em termos da *potência sonora média*, geralmente é apresentada na forma de um gráfico, com a freqüência pura nas abscissas e a potência nas ordenadas.

Existem diversos algoritmos para realizar essa análise em um computador. O mais conhecido é o *Fast Fourier Transform* (FFT, Transformada Rápida de Fourier).


Voltagem
limiar
0.01
0.02

que acaba de gravar. Ao contrário do Spectrum, o TRS-Color executa a melodia na velocidade original. Se você quiser incorporar esse som a um outro programa, poderá gravá-lo em fita com:

```
CSAVEM'MUSICA.',1536,13823,1536
```

Se estiver usando disquetes, adicione 1536 a estes valores. Para gravar a rotina em código que executa a música, utilize:

```
CSAVEM 'TOCA',31000,31174,31091
```

É necessário proteger o topo da memória com **CLEAR 200,30099** e **PCLEAR 8.** Para executar o som, use **EXEC 31091.**

Depois de tudo isso, deve restar muito pouco espaço para seu programa; use-o economicamente.

```
10 PCLEAR 8:CLEAR 200,30999:CLS
:B=191
20 K=31000
30 READ A:IF A<0 THEN 60
40 IF A<256 THEN POKE K,A:K=K+1
:T=T+A:GOTO 30
50 IF T<>A THEN PRINT "erro  VE
RIFIQUE A LINHA";1000+10*INT((K
-31001)/20):END ELSE T=0:GOTO30
60 DEFUSR0=31000:DEFUSR1=31044:
DEFUSR2=31091
70 PRINT @14,"menu":PRINT @131,
"1- GRAFICO DE BARRAS":PRINT @1
95,"2- GRAVAR UM TRECHO":PRINT
@259,"3- REPRODUZIR MUSICA GRAV
ADA"
80 A$=INKEY$:IF A$<"1" OR A$>"3
" THEN 80
90 ON VAL(A$) GOSUB 200,400,600
100 CLS:GOTO 70
200 PMODE 4:COLOR 0,5·CLS
210 PRINT" POSICIONE O GRAVADOR
, APERTE    PLAY E TECLE <ENTER
>"
220 MOTOR ON:AUDIO ON
230 IF INKEY$<>CHR$(13) THEN 23
0
240 MOTOR OFF:PRINT @192,"PRESS
IONE 'M' PARA MENU OU      [SHI
FT] @ PARA CONGELAR IMAGEM"
250 FOR G=1TO 4000:NEXT:MOTORON
260 SCREEN 1,1
270 PCLS:FOR X=0 TO 255:A=B-4*U
SR0(0):IF A<0 THEN A=0
280 LINE (X,B)-(X,A),PSET
290 IF INKEY$="M" THEN X=255:NE
XT:MOTOR OFF:RETURN
300 NEXT:GOTO 270
```

Um sinal analógico que entra pela porta do gravador cassete é lido 2000 vezes por segundo. Ao ultrapassar a voltagem limiar, o sinal é detectado e registrado como 1. Os sinais abaixo desse nível de tensão são registrados como 0. O traçado gráfico digital produzido imita a onda sonora analógica.

MICRO DICAS

APLICAÇÃO EM JOGOS

O programa listado nesse artigo pode tornar seus jogos mais interessantes. Como ele nos permite gravar na memória a imitação de qualquer som de entrada, fica fácil usar a reprodução desse som em um jogo.

Podemos, por exemplo, acrescentar muita emoção a uma aventura introduzindo no programa ruídos de explosões, tiros e até uma mensagem curta de parabéns ao jogador. Digitalizando os sons com o auxílio de nosso programa, eles serão "tocados" pelo alto-falante do micro no momento certo. Você há de concordar que, com esse truque, a qualidade do jogo será incomparavelmente melhor.

```
400 CLS:PMODE3:MOTOR ON·AUDIO O
N:PRINT" POSICIONE O GRAVADOR,
APERTE    PLAY E TECLE <ENTER>"
410 IF INKEY$<>CHR$(13)THEN 410
420 SCREEN 1,0:N=USR1(0)
430 MOTOR OFF:RETURN
600 CLS:PRINT"  REPRODUZINDO TR
ECHO GRAVADO"
610 N=USR2(0)
620 PRINT @129,"NOVAMENTE (S/N)
 ?"
630 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 630
640 IF A$="S" THEN 600
650 RETURN
1000 DATA 26,80,206,255,32,142,
2,233,204,0,0,102,196,37,10,16,
163,132,48,31,1915
1010 DATA 38,245,126,180,244,19
5,0,1,32,7,102,196,36,237,16,16
3,132,48,31,38,2067
1020 DATA 245,126,180,244,26,80
,142,0,0,48,31,38,252,220,25,13
1,0,1,52,6,1847
1030 DATA 158,186,198,8,134,11,
74,38,253,118,255,32,105,132,90
,39,4,18,18,32,1903
1040 DATA 4,48,1,198,8,172,228,
38,231,53,134,26,80,182,255,1,1
32,247,183,255,2476
1050 DATA 1,182,255,3,132,247,1
83,255,3,182,255,35,138,8,183,2
55,35,158,186,220,2916
1060 DATA 25,131,0,1,52,6,134,8
,52,2,230,128,88,36,4,134,252,3
2,3,79,1397
1070 DATA 33,253,183,255,32,134
,8,74,38,253,33,251,106,228,39,
8,109,159,31,64,2291
1080 DATA 30,136,32,224,134,8,1
67,228,172,97,38,214,53,146,167
9,-1
```

INTELIGÊNCIA MILITAR

Veja como a heurística pode

nento, não oferece maiores dificulda-

abuleiro, a superioridade do homem so-

nente dos movimentos do jogador.

Quando a vitória fácil começa a nos
ntediar, a saída é fortalecer nosso opo-
ente. Isso significa incluir mais rotinas.
omo a principal dificuldade na progra-
nação de jogos do tipo *Capa e Espada*

TK-2000 —, tudo o que se adicionar
o programa deve reunir simplicidade e

UM INIMIGO MAIS FORTE

INTELIGÊNCIA MILITAR

Um oponente com forças iguais e

é intelectualmente incapaz. Contudo,

Os conceitos utilizados na programa-
ção da inteligência em *Capa e Espada*
são bem diferentes daqueles que vimos

mentos são muito bem definidos. Em

turos bem como critérios exatos de su-



dependentes, outras não. E, acima de tudo, não há receita para a vitória: nenhum procedimento leva inevitavelmente ao triunfo.

Poderíamos examinar a possibilidade de definir um algoritmo eficaz para o nosso jogo. Mas este seria, de qualquer maneira, extremamente complexo, e tornaria o programa grande e lento demais. Na prática, a saída para programas grandes, complicados e sem algoritmos definidos é dada por um conjunto de *heurísticas*.

Uma heurística é apenas uma regra prática que parece funcionar na maioria dos casos. Não há garantias de que ela funcione nem de que não leve a erros brutais em determinadas situações. Geralmente, no entanto, vale a pena tentar; afinal, este é o procedimento da maior parte das pessoas diante de muitos problemas complicados.

O programa conterá, então, uma lista de regras práticas, a saber:

- Se você é uma unidade de arqueiros e existe uma unidade inimiga ao seu alcance, atire.

- Se você se envolver em uma batalha e estiver perdendo, tente ir em direção contrária.

- Se você está perto de uma unidade inimiga mais forte, afaste-se.

- Se você está próximo a uma unidade inimiga mais fraca, aproxime-se.

No último caso, o computador leva vantagem, já que conhece o poder das unidades do jogador. Este, ao contrário, não tem acesso à mesma informação sobre as tropas inimigas.

CONDUTA PLANEJADA

Além dessas regras, o computador precisa de um plano geral de conduta. O desenrolar de uma guerra nunca é deixado ao acaso. O plano dependerá das condições de vitória — se esta depende, por exemplo, da morte do comandante, poderá ser conveniente usar todas as forças contra o quartel-general. Como em *Capa e Espada* a vitória depende de causar mais baixas ao inimigo do que sofrê-las, o plano do computador terá esse objetivo.

Um modo simples de eliminar as tropas inimigas consistiria em atacar unidades fracas com unidades fortes. Mas esta seria uma estratégia difícil de programar. Ela poderia ser simplificada para "concentre todas as suas forças numa posição". Fazendo isso, as forças do computador superariam as do jogador — desde que este não tivesse planejado a mesma coisa. Essa estratégia, porém, tem uma conseqüência: as forças do jogador superariam as do computador em outra parte do tabuleiro — e aqui a heurística falharia.

Para que o jogador não possa prever a ação do inimigo, o computador deve ter algumas opções abertas. Novamente preservando a simplicidade, podemos fazer a máquina escolher sempre, para concentrar suas forças, uma das posições ocupadas por unidades do jogador. Porém, uma rotina verificaria essas posições, transferindo a concentração ao acaso, a cada volta do laço principal. Isso manteria o jogador na incerteza, ao mesmo tempo que o plano do computador seguiria seu curso.

Um segundo aspecto do jogo a ser inteligentemente controlado é a ação individual das unidades. Cada uma delas deve ser capaz de responder às condições locais do campo de batalha, independentemente do plano geral. Uma unidade não deve, por exemplo, dirigir-se ao ponto de concentração se há unidades inimigas mais fortes no caminho.

Quando o computador tiver que dar ordens, ele fará uma série de testes, numa sequência cuidadosamente escolhida. Essa escolha obedece a certas regras, elaboradas conforme dois critérios. Primeiro: como os testes precisam ser executados com a maior rapidez possível, os que não se aplicam à situação devem ser descartados. Segundo: os testes mais importantes serão realizados no início, de maneira a impedir que uma decisão seja tomada com base em um fator menos relevante.

As regras usadas no programa são as seguintes, por ordem de importância:

- Se a unidade está em combate e venceu na última vez, as ordens são mantidas. Ignoram-se as outras condições.

- Se a unidade está em combate e perdeu na última vez, deve se afastar do inimigo. Isso nem sempre é possível, pois ela pode estar na borda do mapa ou ter o caminho de fuga obstruído.

- Se uma unidade de arqueiros tem um alvo ao alcance, ela deve atirar. A posição dessa regra assegura que os arqueiros prefiram atirar a se envolver em combate corpo a corpo.

- Se existe uma unidade inimiga mais fraca a uma distância inferior ao deslocamento máximo, MARCHE em direção a ela. Entretanto, se a unidade for mais forte, afaste-se. Este é um teste demorado, uma vez que leva em conta as unidades inimigas.

- Dirija-se ao ponto de concentração.

Mesmo essas poucas regras levam um bom tempo para serem consideradas, diminuindo bastante a velocidade do jogo. Este é o preço a ser pago por um jogo inteligente. Não haverá, porém, esperas tão longas e exasperantes como em *A Raposa e os Gansos*.

ROTINAS ADICIONAIS

Apague as linhas 1770 a 1790 antes de adicionar estas rotinas.



```
 360 DIM t$(8,12): DIM o$(5,12)
 : DIM w$(5,9): DIM m$(5,12):
 DIM a$(4,12): DIM r$(4,12):
 DIM c(8)
 416 LET sp=1
1665 IF wn>8 THEN  LET c(wn-8)=
8
1666 IF lo>8 THEN  LET c(lo-8)=
T(wn,2)
1760 LET ra=st: LET rb=sh  LET
rc=fx: LET rd=fy: GOSUB 3000
2140 REM Inimigo
2142 REM Loop
2143 LET r=FN r(10)
2144 IF r=1 OR T(sp,1)>3 THEN
LET sp=FN r(8)
2145 IF r=1 AND T(sp,1)>3 THEN
 GOTO 2142
2150 IF c(e-8)=8 THEN  RETURN
2155 IF c(e-8)<>0 THEN  LET T(e
,1)=3: LET T(e,2)=c(e-8): LET c
(e-8)=0: RETURN
2170 IF T(e,3)=2 THEN  LET ra=1
: LET rb=e: LET rc=5: LET rd=5:
 GOSUB 3000: IF gp<>-1 THEN  LE
T T(e,1)=1: RETURN
2180 LET T(e,1)=3
2181 LET hp=5: LET vp=5: LET mv
=0
2182 FOR v=1 TO 8
2183 LET zp=0: GOSUB 3100
2184 IF zp<>0 THEN  GOSUB 3200
2185 NEXT v
2187 IF hp<>5 AND vp<>5 THEN  R
ETURN
2188 IF mv<>0 THEN  LET T(e,2)=
mv: RETURN
2189 LET hp=T(e,8)-T(sp,8): LET
 vp=T(e,9)-T(sp,9): GOSUB 3200
2190 RETURN
3000 REM
3010 LET gp=-1
3020 FOR m=ra TO (ra+7)
3030 LET xx=ABS (T(m,8)-T(rb,8)
): LET yy=ABS (T(m,9)-T(rb,9))
3040 IF xx<rc AND yy<rd AND T(m
,1)<4 THEN  LET rc=xx: LET rd=y
y: LET gp=m
```

```
3050 NEXT m
3060 RETURN
3100 REM
3110 IF T(v,1)>3 THEN  RETURN
3120 LET xx=ABS (T(v,8)-T(e,8))
: LET yy=ABS (T(v,9)-T(e,9))
3130 IF T(v,7)>=T(e,7) AND xx<5
 AND yy<5 THEN  LET mv=T(v,2)
3140 IF xx<hp AND yy<vp THEN  L
ET hp=xx: LET vp=yy: LET zp=1:
RETURN
3150 RETURN
3200 REM
3210 IF hp>=0 THEN  LET lp=1
3220 IF hp<0 THEN  LET lp=3: LE
T hp=ABS (hp)
3230 IF vp>=0 AND ABS (vp)>hp T
HEN  LET lp=2
3240 IF vp<0 AND ABS (vp)>hp TH
EN  LET lp=4· LET vp=ABS (vp)
3250 LET T(e,2)=lp
3260 RETURN
```



```
360 DIM T$(8),O$(5),W$(5),M$(5)
,A$(4),R$(4),C(8)
416 SP=1
1665 IF WN>8 THEN C(WN-8)=8
1666 IF LO>8 THEN C(LO-8)=T(WN,
2)
1760 RA=ST:RB=SH:RC=FX:RD=FY:GO
SUB 3000
2140 REM  INIMIGO
2142 REM  LOOP
2143 R=RND(10)
2144 IF R=1 OR T(SP,1)>3 THEN S
P=RND(8)
2145 IF R=1 AND T(SP,1)>3 THEN
2142
2150 IF C(E-8)=8 THEN RETURN
2155 IF C(E-8)<>0 THEN T(E,1)=3
:T(E,2)=C(E-8):C(E-8)=0:RETURN
2170 IF T(E,3)=2 THEN RA=1:RB=E
:RC=5:RD=5:GOSUB 3000:IF GP<>-1
 THEN T(E,1)=1:RETURN
2180 T(E,1)=3
2181 HP=5:VP=5:MV=0
2182 FOR V=1 TO 8
2183 ZP=0:GOSUB 3100
2184 IF ZP<>0 THEN GOSUB 3200
2185 NEXT V
2187 IF HP<>5 AND VP<>5 THEN RE
TURN
2188 IF MV<>0 THEN T(E,2)=MV:RE
TURN
2189 HP=T(E,8)-T(SP,8):VP=T(E,9
)-T(SP,9).GOSUB 3200
2190 RETURN
3000 REM
3010 GP=-1
3020 FOR M=RA TO (RA+7)
3030 XX=ABS(T(M,8)-T(RB,8)):YY=
ABS(T(M,9)-T(RB,9))
3040 IF XX<RC AND YY<RD AND T(M
,1)<4 THEN RC=XX:RD=YY:GP=M
3050 NEXT M
3060 RETURN
3100 REM
3110 IF T(V,1)>3 THEN RETURN
3120 XX=ABS(T(V,8)-T(E,8)):YY=A.
```

```
BS(T(V,9)-T(E,9))
3130 IF T(V,7)>=T(E,7) AND XX<5
 AND YY<5 THEN MV=T(V,2)
3140 IF XX<HP AND YY<VP THEN HP
=XX:VP=YY:ZP=1:RETURN
3150 RETURN
3200 REM
3210 IF HP>=0 THEN LP=1
3220 IF HP<0 THEN LP=3:HP=ABS(H
P)
3230 IF VP>=0 AND ABS(VP)>HP TH
EN LP=2
3240 IF VP<0 AND ABS(VP)>HP THE
N LP=4:VP=ABS(VP)
3250 T(E,2)=LP
3260 RETURN
```



```
360  DIM T$(8),O$(5),W$(5),M$(5
),A$(4),R$(4),C(8)
416 SP = 0
1665  IF WN > 8 THEN C(WN - 8)
= 8
1666  IF LO > 8 THEN C(LO - 8)
= T(WN,2)
1760 RA = ST:RB = SH:RC = FX:RD
 = FY: GOSUB 3000
2140  REM  SELECAO
2143 R =  FN R(10)
2144  IF R = 1 OR T(SP,1) > 3 T
HEN SP =  FN R(8)
2145  IF R = 1 AND T(SP,1) > 3
THEN 2143
2150  IF C(E - 8) = 8 THEN  RET
URN
2155  IF C(E - 8) <  > 0 THEN T
(E,1) = 3:T(E,2) = C(E - 8):C(E
 - 8) = 0: RETURN
2170  IF T(E,3) = 2 THEN RA = 1
:RB = E:RC = 5:RD = 5: GOSUB 30
00: IF GP <  >  - 1 THEN T(E,1)
 = 1: RETURN
2180 T(E,1) = 3
2181 HP = 5:VP = 5:MV = 0
2182  FOR V = 1 TO 8
2183 ZP = 0: GOSUB 3100
2184  IF ZP <  > 0 THEN  THEN
GOSUB 3200
2185  NEXT V
2187  IF HP <  > 5 AND VP <  >
5 THEN  RETURN
2188  IF MV <  > 0 THEN T(E,2)
= MV: RETURN
2189 HP = T(E,8) - T(SP,8):VP =
 T(E,9) - T(SP,9): GOSUB 3200
2190  RETURN
3000  REM  ALCANCE
3010 GP =  - 1
3020  FOR M = RA TO (RA + 7)
3030 XX =  ABS (T(M,8) - T(RB,8
)):YY =  ABS (T(M,9) - T(RB,9))
3040  IF XX < RC AND YY < RD AN
D T(M,1) < 4 THEN RC = XX:RD =
YY:GP = M
3050  NEXT M
3060  RETURN
3100  REM  PODER
3110  IF T(V,1) > 3 THEN  RETUR
N
3120 XX =  ABS (T(V,8) - T(E,8)
):YY = T(V,9) - T(E,9))
3130  IF T(V,7) >  = T(E,7) AND
 XX < 5 AND YY < 5 THEN MV = T(
V,2)
3140  IF XX < HP AND YY < VP TH
EN HP = XX:VP = YY:ZP = 1: RETU
RN
3150  RETURN
3200  REM  CONCENTRACAO
3210  IF HP >  = 0 THEN LP = 1
3220  IF HP < 0 THEN LP = 3:HP
=  ABS (HP)
3230  IF VP >  = 0 AND  ABS (VP
) > HP THEN LP = 2
3240  IF VP < 0 AND  ABS (VP) >
 HP THEN LP = 4:VP =  ABS (VP)
3250 T(E,2) = LP
3260  RETURN
```



As rotinas que começam em 3000 devem ser renumeradas a fim de liberar espaço para essas linhas. Digite:

```
RENUM 4000,3000
```

Em seguida, acrescente estas linhas:

```
360 DIM T$(8),O$(5),W$(5),M$(5)
,A$(4),R$(4),C(8)
416 SP=1
1665 IF WN>8 THEN C(WN-8)=8
1666 IF LO>8 THEN C(LO-8)=T(WN,
2)
1760 RA=ST:RB=SH:RC=FX:RD=FY:GO
SUB 3000
2140 REM  INIMIGO
2142 REM  LOOP
2143 R=RND(10)
2144 IF R=1 OR T(SP,1)>3 THEN S
P=RND(8)
2145 IF R=1 AND T(SP,1)>3 THEN
2142
2150 IF C(E-8)=8 THEN RETURN
2155 IF C(E-8)<>0 THEN T(E,1)=3
:T(E,2)=C(E-8):C(E 8)=0:RETURN
2170 IF T(E,3)=2 THEN RA=1:RB=E
:RC=5:RD=5:GOSUB 3000:IF GP<>-1
 THEN T(E,1)=1:RETURN
2180 T(E,1)=3
2181 HP=5:VP=5:MV=0
2182 FOR V=1 TO 8
2183 ZP=0:GOSUB 3100
2184 IF ZP<>0 THEN GOSUB 3200
2185 NEXT V
2187 IF HP<>5 AND VP<>5 THEN RE
TURN
2188 IF MV<>0 THEN T(E,2)=MV:RE
TURN
2189 HP=T(E,8)-T(SP,8):VP=T(E,9
)-T(SP,9):GOSUB 3200
2190 RETURN
3000 REM
3010 GP=-1
3020 FOR M=RA TO (RA+7)
3030 XX=ABS(T(M,8)-T(RB,8)):YY=
ABS(T(M,9)-T(RB,9))
3040 IF XX<RC AND YY<RD AND T(M
,1)<4 THEN RC=XX:RD=YY:GP=M
3050 NEXT M
3060 RETURN
3100 REM
3110 IF T(V,1)>3 THEN RETURN
3120 XX=ABS(T(V,8)-T(E,8)):YY=A
BS(T(V,9)-T(E,9))
3130 IF T(V,7)>=T(E,7) AND XX<5
 AND YY<5 THEN MV=T(V,2)
3140 IF XX<HP AND YY<VP THEN HP
=XX:VP=YY:ZP=1:RETURN
3150 RETURN
3200 REM
3210 IF HP>=0 THEN LP=1
3220 IF HP<0 THEN LP=3:HP=ABS(H
P)
3230 IF VP>=0 AND ABS(VP)>HP TH
EN LP=2
3240 IF VP<0 AND ABS(VP)>HP THE
N LP=4:VP=ABS(VP)
3250 T(E,2)=LP
3260 RETURN
```

COMO FUNCIONA

As adições anteriores à linha 2140 cuidam de algumas variáveis extras que serão utilizadas.

No início da rotina de escolha, há uma chance em dez de que o computador mude o ponto de concentração. Este é verificado e modificado de acordo, mas não se realiza nenhuma ação neste estágio — outros testes precisam ainda ser feitos.

Se a unidade for constituída de arqueiros, a linha 2170 usa a rotina de alcance para decidir se atira. Caso os arqueiros não disparem, ou se a unidade for de outro tipo, a linha 2180 muda a ordem para MARCHE. A seção seguinte do programa verifica cada uma das unidades inimigas. A linha 2183 utiliza a rotina de poder para determinar a direção do movimento.

Se a unidade em questão não estiver envolvida em combate nem prestes a se envolver, a rotina de concentração (linha 3500) faz com que ela se mova em direção ao ponto de concentração.

A ESCOLHA É SUA

Você pode adicionar qualquer das versões aqui apresentadas, ou mesmo as duas, para melhorar o desempenho do micro, enquanto jogador. Tudo dependerá do tipo de oponente que desejar.

A própria natureza das heurísticas incorporadas ao programa faz com que elas falhem em certas circunstâncias. Alguns jogadores acharão essas régras menos adequadas do que outras eventualmente deduzidas pela prática. Só um processo de tentativa e erro determinará as melhores estratégias e muitas outras rotinas poderão ser tentadas.

NOVAS MENSAGENS SECRETAS

- COMO DECIFRAR CÓDIGOS
- PROGRAMA DE DISTRIBUIÇÃO DE LETRAS
- CÓDIGOS MULTIPLICATIVOS
- LIVRO-CÓDIGO

Neste artigo você ficará conhecendo novas maneiras de transmitir mensagens confidenciais, sem correr o risco de que elas sejam decifradas por pessoas indesejadas.

No artigo *Mensagens Secretas* (página 888), sugerimos vários métodos para a transmissão de informações de caráter confidencial. Alguns deles são relativamente fáceis de serem decifrados; entretanto, se recorrermos ao auxílio do computador, poderemos chegar a métodos mais complexos e com diferentes níveis de sofisticação.

COMO DECIFRAR CÓDIGOS

Da mesma maneira que os criptógrafos tentam desenvolver códigos que ofereçam mais segurança, muitos especialistas estudam uma forma de frustrar essas tentativas. Uma arma muito poderosa para se decifrar simples códigos de substituição ou de transposição consiste na contagem de frequência das letras. Em português, as letras que ocorrem com mais frequência são — nesta ordem — A, E, O, S, R, I, C. Desse modo, se em um texto codificado aparecerem muitas vezes as letras AEOSRIC, provavelmente você estará manipulando um código de transposição. Se outras letras se repetirem mais vezes, a codificação pode ter sido feita com um método de substituição. É importante levar em conta que essa distribuição é exclusiva de cada idioma. Portanto, torna-se fundamental saber em que língua foi escrita a mensagem.

Seja como for, será necessário fazer a contagem da frequência das letras na mensagem. Isso pode consumir muito tempo e é um processo bastante susceptível de erros. O programa aqui apresentado lhe será útil nesses casos. Basta que você digite o texto e o computador imediatamente o informará do número de vezes que cada letra apareceu.

A figura da página 1091 mostra a frequência das letras em um texto consti-

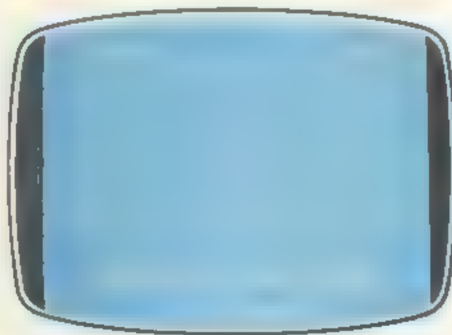


tuído de cem palavras. Observe que os números seguem, razoavelmente, o princípio que acabamos de comentar.

Agora, digite o programa e veja como isso funciona na prática.

```
 15 POKE 23658,8
 20 BORDER 0: PAPER 0: INK 7:
CLS
 30 PRINT TAB 5;"CONTAGEM DA F
REQUENCIA"''
 40 PRINT TAB 12;"CUIDADO"''
 50 PRINT  FLASH 1;AT 4,7;"NAO
  DEIXE ESPACOS";AT 5,9;"ENTR
E PALAVRAS"''
 60 DIM n(28)
 70 PRINT " Para finalizar ent
rada do texto digite '*'"
 80 FOR t=1 TO 28: LET n(t)=0:
 NEXT t
 90 INPUT "Introduza o texto";
 a$
100 IF a$="*" THEN  GOTO 180
110 CLS
120 FOR i=1 TO LEN a$
130 FOR j=1 TO 26
140 IF j=CODE (a$(i TO i))-64
THEN  LET n(j)=n(j)+1
150 NEXT j
160 NEXT i
170 GOTO 90
180 CLS
190 PRINT "Letra    Freq'   Le
tra   Freq'"
200 FOR i=1 TO 13
210 PRINT TAB 2;CHR$ (64+i);
TAB 10;n(i);TAB 19;CHR$ (77+i)
;TAB 27;n(13+i)
220 NEXT i
230 STOP
```

```
20 CLS
30 PRINT @5,"CONTAGEM DA FREQUE
NCIA"
40 PRINT @76,"CUIDADO"
50 PRINT @134,"NAO DEIXE ESPACO
S":PRINT @166,"ENTRE AS PALAVRA
S"
60 DIM N(28)
70 PRINT @225,"PARA FINALIZAR E
NTRADA DO TEXTO E EMITIR RESULT
ADOS DIGITE *"
80 FOR T=1 TO 28:N(T)=0:NEXT
90 INPUT"INTRODUZA O TEXTO ";A$
100 IF A$="*" THEN 180
110 CLS
120 FOR I=1 TO LEN(A$)
130 FOR J=1 TO 26
140 IF J=ASC(MID$(A$,I,1))-64 T
HEN N(J)=N(J)+1
150 NEXT J
160 NEXT I
170 GOTO 90
180 CLS
190 PRINT "LETRA    FREQ'   LET
RA    FREQ'"
200 FOR I=1 TO 13
210 PRINT TAB(2);CHR$(64+I);TAB
(10),N(I),TAB(19);CHR$(77+I);TA
B(27);N(13+I)
220 NEXT
230 END
```



A tabela de frequência das letras tem grande utilidade na decifração de códigos.



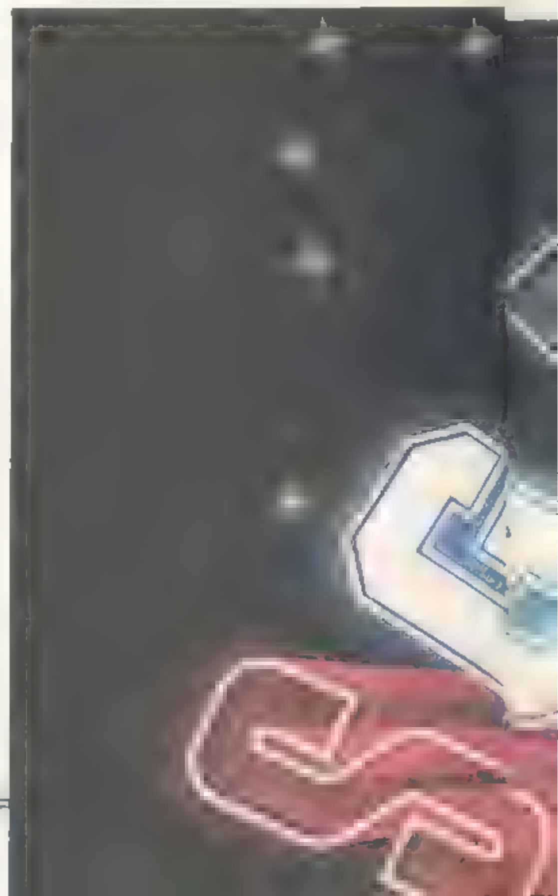
```
20  HOME
30  PRINT  TAB( 9)"CONTAGEM DE
FREQUENCIA": PRINT
40  PRINT  TAB( 17)"CUIDADO": P
RINT
50  PRINT  TAB( 12)"NAO DEIXE E
SPACOS": PRINT  TAB( 12)"ENTRE
AS PALAVRAS"
60  DIM N(28)
70  PRINT : PRINT : PRINT  TAB(
 8)"PARA FINALIZAR A ENTRADA":
PRINT  TAB( 11)" DO TEXTO DIGIT
E *"
80  FOR I = 1 TO 28:N(T) = 0: N
EXT
90  PRINT : INPUT "DIGITE O TEX
TO ";A$
100  IF A$ = "*" THEN 180
110  HOME
120  FOR I = 1 TO  LEN (A$)
130  FOR J = 1 TO 26
140  IF J =  ASC ( MID$ (A$,I,1
)) - 64 THEN N(J) = N(J) + 1
150  NEXT J
160  NEXT I
170  GOTO 90
180  HOME
190  PRINT "LETRA   FREQ   LETR
A   FREQ"
200  FOR I = 1 TO 13
210  PRINT  TAB( 2); CHR$ (64 +
 I); TAB( 10);N(I); TAB( 17); C
HR$ (77 + I); TAB( 25);N(13 + I
)
220  NEXT I
230  END
```



```
20 CLS
30 PRINT TAB(9)"CONTAGEM DE FRE
QUENCIA":PRINT
40 PRINT TAB(17)"CUIDADO":PRINT
50 PRINT TAB(12)"NAO DEIXE ESPA
ÇOS":PRINT TAB(12)"ENTRE AS PAL
AVRAS"
60 DIM N(28)
70 PRINT:PRINT:PRINT TAB(8)"PAR
A FINALIZAR A ENTRADA":PRINT TA
B(12)"DO TEXTO DIGITE *"
80 FOR I=1 TO 28:N(T)=0:NEXT
90 PRINT:INPUT "DIGITE O TEXTO"
;A$
100 IF A$="*" THEN 180
110 CLS
120 FOR I=1 TO LEN(A$)
130 FOR J=1 TO 26
140 IF J=ASC(MID$(A$,I,1))-64 T
HEN N(J)=N(J)+1
150 NEXT J
160 NEXT I
170 GOTO 90
180 CLS
190 PRINT "LETRA   FREQ   LETRA
  FREQ"
200 FOR I=1 TO 13
210 PRINT TAB(2);CHR$(64+I);TAB
(10);N(I);TAB(17);CHR$(77+I);TA
B(25);N(13+I)
220 NEXT I
230 END
```

A estrutura operacional deste programa está fundamentada em um simples mecanismo de contagem.

Em sua primeira parte é atribuído o valor zero às 28 variáveis indexadas, sendo que 26 dessas variáveis serão usadas para a contagem da frequência com que aparecem as 26 letras. As duas variáveis restantes foram incluídas tendo em vista a possibilidade de uma futura ampliação do programa; nesse caso, podem ser incorporados algarismos ou quaisquer outros sinais gráficos.



CÓDIGOS MULTIPLICATIVOS

Durante a Guerra Civil Norte-Americana (1861-1865) criou-se um tipo especial de código para a comunicação entre as tropas, que funciona da seguinte forma: suponhamos que se queira passar para um oficial preso a mensagem ESCAPAR PARA LONDRES. Nesse caso, a primeira letra da frase seria colocada na primeira linha, a segunda letra na segunda linha, a terceira letra de volta na primeira linha e assim por diante, de tal maneira que a frase acabaria ficando assim:

E C P R A A O D E
S A A P R L N R S

Escrita por extenso, a mensagem seria a seguinte: ECPRAAODESAAPRLNRS; dividida a fim de confundir o inimigo: ECPRAA OD ESAAP RLNRS.

Na verdade, esse código é um caso especial do que se chama atualmente código multiplicativo.

Em nossa mensagem, o texto contém dezoito caracteres. Estes podem ser arranjados em matrizes do tipo: 2 x 9, 9 x 2, 3 x 6, ou 6 x 3, como mostram as figuras da página 1095.

Qualquer pessoa que queira decifrar a mensagem, sem saber que se trata de um código multiplicativo, terá muito trabalho. O processo de codificação, no entanto, é muito simples: tudo o que se tem a fazer é escrever a mensagem verticalmente, letra por letra, preenchendo totalmente a primeira coluna da matriz. Em seguida, partimos para a próxima coluna e repetimos o processo, até que a mensagem termine e a matriz esteja completa. Finalmente, copiamos as linhas lado a lado, começando sempre pela primeira.

É importante que a mensagem ocupe totalmente a matriz escolhida. Para tanto, podemos introduzir no texto alguns caracteres ou palavras sem nexo, que terão a função extra de confundir quem tentar decifrá-lo.

O laço que se encontra entre as linhas 140 e 210 é a parte principal do programa, responsável tanto pela codificação como pela decodificação.

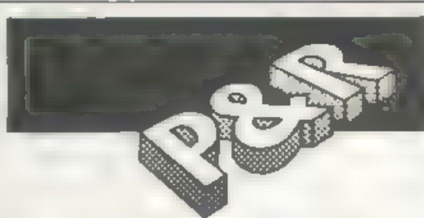
S

```
20 BORDER 0: PAPER 0: INK 7:
CLS
30 PRINT TAB (6);"CODIGO MULT
IPLICATIVO"
40 PRINT : PRINT : PRINT
50 PRINT  FLASH 1; PAPER 2;"N
AO DEIXE ESPACOS ENTRE PALAVR
AS"
60 INPUT "INTRODUZA O TEXTO "
'm$
70 INPUT "LINHAS ?  ";m
80 INPUT "COLUNAS ? ";n
90 INPUT "(c)ODIFICAR OU (d)E
CODIFICAR ? ";e$
100 PAUSE 50: CLS
110 IF e$="c" THEN  LET x=m
120 IF e$="d" THEN  LET x=n
130 DIM d$(x,LEN m$/x)
140 FOR i=1 TO x
155 LET a$="": LET m$=m$+" "
160 FOR j=1 TO LEN m$-1 STEP x
180 LET a$=a$+m$(i+j-1 TO i+j-
1)
190 NEXT j
195 LET d$(i)=a$
197 LET m$=m$( TO LEN m$-1)
200 PRINT d$(i);: IF e$="c"
THEN  PRINT " ";
210 NEXT i
220 STOP
```

T

```
20 CLS
30 PRINT @6,"CODIGO MULTIPLICAT
IVO"
40 PRINT:PRINT:PRINT
50 PRINT"NAO DEIXE ESPACOS ENTR
E PALAVRAS"
60 PRINT:INPUT"TEXTO  ";M$
70 INPUT"LINHAS ";M
80 INPUT"COLUNAS ";N
90 INPUT"(C)ODIFICAR OU (D)ECOD
IFICAR ";E$
100 FOR L=1 TO 1000:NEXT
110 IF E$="C" THEN X=M
120 IF E$="D" THEN X=N
130 DIM D$(X)
140 FOR I=1 TO X
150 D$(I)=""
160 FOR J=1 TO LEN(M$) STEP X
170 B$=MID$(M$,I+J-1,1)
180 D$(I)=D$(I)+B$
190 NEXT J
200 PRINT D$(I)
210 NEXT I
220 END
```





Há vantagens em se criptografar um programa de computador?

A resposta depende do tipo de programa. Não há nenhuma vantagem em criptografar um programa em linguagem de máquina (código binário) — tarefa, aliás, impossível para muitos micros, que não poderão executá-lo depois. Mas, se o programa estiver em código fonte (linguagem de alto nível, como BASIC), às vezes pode ser vantajoso criptografá-lo.

O objetivo da criptografia de programas é protegê-los contra a cópia e a imitação ilegais. Uma aplicação muito freqüente ocorre na transmissão de programas de computador em sistemas telemáticos. Aqui, o objetivo é torná-los imunes à cópia ilegal no momento da transmissão. Nesse caso, o interessado pode criptografar o programa como um texto qualquer, usando um dos métodos explicados neste artigo (o multiplicativo costuma ser o mais empregado).

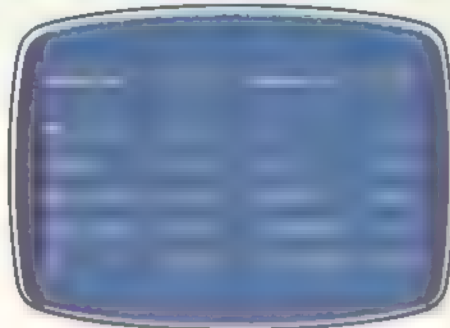


```
20   HOME
30   PRINT  TAB( 10)"CODIGO MULT
IPLICATIVO"
40   PRINT : PRINT : PRINT
50   PRINT  TAB( 12)"NAO DEIXE E
SPACOS": PRINT  TAB( 12)"ENTRE
AS PALAVRAS"
60   PRINT : INPUT "TEXTO:";M$
70   INPUT "LINHAS:";M
80   INPUT "COLUNAS:";N
90   INPUT "CODIFICAR(C) OU DECO
DIFICAR(D):";E$
100   FOR L = 1 TO 1000: NEXT
110   IF E$ = "C" THEN X = M
120   IF E$ = "D" THEN X = N
130   DIM D$(X)
140   FOR I = 1 TO X
150   IF E$ = "C" THEN D$(I) = "
"
160   FOR J = 1 TO  LEN (M$) STE
P X
170 B$ =  MID$ (M$,I + J - 1,1)

180 D$(I) = D$(I) + B$
190   NEXT J
200   PRINT D$(I);: IF E$ = "C"
THEN  PRINT
210   NEXT I
220   END
```



```
20 CLS
30 PRINT TAB(10)"CODIGO MULTIPL
ICATIVO"
40 PRINT:PRINT:PRINT
50 PRINT TAB(12)"NAO DEIXE ESPA
COS":PRINT TAB(12)"ENTRE AS PAL
AVRAS"
60 PRINT:INPUT"TEXTO:";M$
70 INPUT"LINHAS:",M
80 INPUT"COLUNAS:";N
90 INPUT"CODIFICAR(C) OU DECODI
FICAR(D):";E$
100 FOR L=1 TO 1000:NEXT
110 IF E$="C" THEN X=M
120 IF E$="D" THEN X=N
130 DIM D$(X)
140 FOR I=1 TO X
150 IF E$="C" THEN D$(I)=" "
160 FOR J=1 TO LEN(M$) STEP X
170 B$=MID$(M$,I+J-1,1)
180 D$(I)=D$(I)+B$
190 NEXT J
200 PRINT D$(I);:IF E$="C" THEN
 PRINT
210 NEXT I
220 END
```

Para usar o sistema de livro-código, tanto o emissor quanto o receptor da mensagem precisam ter uma cópia idêntica do dicionário fixo.

LIVRO-CÓDIGO

Até agora, trabalhamos apenas com cifras. Os códigos propriamente ditos consistem na substituição de uma palavra ou frase por outras. Isso exige que se recorra sempre a um livro-código, que deve ser guardado com muito cuidado, de modo a evitar que caia em mãos estranhas. Por esse motivo, geralmente, os textos são codificados e recebidos em lugares fixos, bem seguros.

O programa seguinte monta um dicionário de códigos de vinte palavras. Usando uma matriz bidimensional **A$(I,J)**, onde I=1 guarda a palavra e I=2 guarda seu código, primeiramente são lidas as informações após os comandos **DATA**. A parte seguinte, que vai da linha 120 à linha 170, recebe a palavra e imprime o código correspondente, ou faz o contrário, caso você tenha optado pela decodificação.

A mensagem FUGIR DE ROMA À MEIA-NOITE DE DOMINGO CHEGAR A NOVA YORK AO MEIO-DIA seria codificada como: 40476 10996 23874 12128 26569 10996 40553 68719 12128 54982 69783 11072. O texto codificado: 74891 22317 69783 74832 seria traduzido para ENVIAR DINHEIRO AO ANOITECER.

Com o próximo programa, você poderá codificar e decodificar rapidamente suas mensagens. Nas situações reais, onde são transmitidos textos muito grandes e os dicionários são volumosos, o computador exerce um papel fundamental, poupando bastante tempo.



```
20 BORDER 0: PAPER 0: INK 7:
CLS
25 POKE 23658,8
30 PRINT TAB (10);"LIVRO DE C
ODIGOS"
40 PRINT : PRINT : PRINT
50 DIM a$(2,20,10)
60 FOR i=1 TO 2
70 FOR j=1 TO 20
80 READ a$(i,j)
90 NEXT j: NEXT i
100 INPUT "CODIFICAR (0) OU DE
CODIFICAR (1)";x
110 CLS
120 INPUT "Digite a palavra";m
$
125 IF LEN m$>=10 THEN  GOTO
130
127 FOR n=1 TO 10-LEN m$: LET
m$=m$+" ": NEXT n
130 IF m$="*" THEN  GOTO 280
140 FOR t=1 TO 20
150 IF m$=a$(1+x,t) THEN
PRINT a$(2-x,t)
160 NEXT t
170 GOTO 120
180 DATA "BRASILIA","LONDRES",
"PARIS","ROMA"
190 DATA "CHEGADA","SAIDA","VA
 PARA","FUJA PARA","SABADO"
200 DATA "DOMINGO","MEIO-DIA",
"AMANHECER","MEIA-NOITE"
210 DATA "ANOITECER","EM","AO"
```





Usando o código de multiplicação com diversas chaves, você poderá criptografar a mesma mensagem de várias maneiras. O texto cifrado é sempre apresentado em grupos de letras de comprimento fixo.

```
,"NO","ENVIE"
220 DATA "DINHEIRO","COMIDA"
230 DATA "54982","73581","9007
5","23874"
240 DATA "68719","68677","1032
7","40476"
250 DATA "27921","48553","1107
2","70355"
260 DATA "26569","74832","1099
6","12128"
270 DATA "69783","74891","2231
7","98724"
280 STOP
```



```
20 CLS
30 PRINT @8,"LIVRO DE CODIGO"
40 PRINT.PRINT.PRINT
50 DIM A$(2,20)
60 FOR I=1 TO 2
70 FOR J=1 TO 20
80 READ A$(I,J)
90 NEXT J,I
100 INPUT"CODIFICAR (0) OU DECO
DIFICAR (1)";X
110 CLS
120 INPUT"DIGITE A PALAVRA ";M$
130 IF M$="*" THEN END
140 FOR T=1 TO 20
150 IF M$=A$(1+X,T) THEN PRINT
A$(2-X,T)
160 NEXT
170 GOTO 120
180 DATA BRASILIA,LONDRES,PARIS
,ROMA
190 DATA CHEGADA,PARTIDA DE,VA
PARA,FUJA PARA,SABADO
200 DATA DOMINGO,MEIO-DIA,AMANH
ECER,MEIA-NOITE
210 DATA ANOITECER,EM,AO,NO,ENV
IE
220 DATA DINHEIRO,COMIDA
230 DATA 54982,73581,90075,2387
4
240 DATA 68719,68677,10327,4047
6
250 DATA 27921,48553,11072,7035
5
260 DATA 26569,74832,10996,1212
8
270 DATA 69783,74891,22317,9872
4
```




```
20  HOME
30  PRINT  TAB( 14)"LIVRO CODIG
O"
40  PRINT : PRINT : PRINT
50  DIM A$(2,20)
60  FOR I = 1 TO 2
70  FOR J = 1 TO 20
80  READ A$(I,J)
90  NEXT J
95  NEXT I
100  INPUT "CODIFICAR(0) OU DEC
ODIFICAR(1) ?";X
110  HOME
120  INPUT "DIGITE A PALAVRA ";
M$
130  IF M$ = "*" THEN  END
140  FOR T = 1 TO 20
150  IF M$ = A$(1 + X,T) THEN
PRINT A$(2 - X,T)
160  NEXT T
170  GOTO 120
180  DATA   NOVAYORK,LONDRES,PA
RIS,ROMA
190  DATA  CHEGAR,PARTIR,IR,FUG
IR,SABADO
200  DATA   DOMINGO,MEIO DIA,TA
RDE,MEIA-NOITE
210  DATA    ANOITECER,DE,A,AO,
ENVIAR
220  DATA   DINHEIRO,SUPRIMENTO

230  DATA  54982,73581,90075,23
874
240  DATA  68719,68677,10327,40
476
250  DATA  27921,48553,11072,70
355
260  DATA  26569,74832,10996,12
128
270  DATA  69783,74891,22317,98
724
```



```
20 CLS
30 PRINT TAB(14)"LIVRO CODIGO"
40 PRINT:PRINT:PRINT
50 DIM A$(2,20)
60 FOR I=1 TO 2
70 FOR J=1 TO 20
80 READ A$(I,J)
90 NEXT J
95 NEXT I
100 INPUT "CODIFICAR(0) OU DECO
DIFICAR(1) ";X
110 CLS
120 INPUT "DIGITE A PALAVRA ";M
$
130 IF M$="*" THEN END
140 FOR T=1 TO 20
150 IF M$=A$(1+X,T) THEN PRINT
A$(2-X,T)
160 NEXT T
170 GOTO 120
180 DATA NOVAYORK,LONDRES,PARIS
,ROMA
190 DATA CHEGAR,PARTIR,IR,FUGIR
,SABADO
200 DATA DOMINGO,MEIO-DIA,TARDE
,MEIA-NOITE
210 DATA ANOITECER,DE,A,AO,ENVI
AR
220 DATA DINHEIRO,SUPRIMENTO
230 DATA 54982,73581,90075,2387
4
240 DATA 68719,68677,10327,4047
6
250 DATA 27921,48553,11072,7035
5
260 DATA 26569,74832,10996,1212
8
270 DATA 69783,74891,22317,9872
4
```

Apesar de estar limitado a apenas vinte palavras, o programa pode ser facilmente ampliado. Por exemplo, se você quiser fazer um dicionário de cinqüenta palavras, bastará trocar 20 por 50 nas linhas 50, 70 e 140. É claro que você terá também que fazer novas linhas **DATA**, com as palavras e códigos suplementares.

No programa atual para o Apple, você não poderá introduzir palavras com mais de dez caracteres. Caso queira ampliar este número para doze, por exemplo, você deve substituir 10 por 12 na linha 50. Essa restrição não existe nos outros programas.

PÁGINAS GRÁFICAS

Utilizando páginas gráficas, você poderá criar as mais diversas figuras e cenas animadas no microcomputador. Elas tornarão seus jogos muito mais divertidos.

Todos os tipos de animação relacionam-se a um fenômeno da percepção conhecido como persistência da visão. Cada imagem transmitida ao cérebro permanece "gravada" na memória por alguns instantes, mesmo que nossa visão já esteja captando uma nova imagem. Porém, quando uma série de imagens é mostrada com muita rapidez, o cérebro não consegue separá-las, pois não processa mais de doze imagens por segundo. As imagens parecem, então, sair umas das outras, e é exatamente isto que dá a impressão de movimento.

Certos livros infantis, cujas ilustrações compõem uma seqüência de imagens, permitem que se perceba esse fenômeno com clareza. Folheando-os rapidamente, suas figuras parecem se movimentar. Também o jogo de sombras chinesas, um dos mais remotos precursores do cinema, vale-se desse fenômeno: figuras recortadas ou criadas com as mãos são projetadas sobre paredes ou telas de linho em ritmo acelerado, formando uma sombra animada.

Finalmente, apesar dos progressos tecnológicos, o próprio cinema recorre ao mesmo princípio básico: os quadros, fixados em um filme, são projetados à velocidade de 25 unidades por segundo. Criam-se, assim, duas ilusões ópticas: a de que há uma corrente contínua de imagens, quando, na verdade, elas se sucedem de modo descontínuo; e a de que coisas imóveis têm movimento.

A montagem de um desenho animado é bem mais trabalhosa. O artista desenha sobre uma folha transparente. Ao mudar de quadro, sobrepõe a essa folha uma outra, e copia o desenho anterior, mudando ligeiramente sua forma. Tente calcular quantas cenas ele precisaria desenhar para produzir um desenho animado de uma hora de duração...

GRÁFICOS NO COMPUTADOR

Por que não usar o computador para agilizar esse processo, se até os micros mais simples são capazes de fazer desenhos de boa qualidade?

A capacidade gráfica dos micros atuais chega a ser impressionante. Entretanto, isso não é suficiente para satisfazer o público de um cinema. O custo da produção de um filme de ficção científica em computador chega a ser da ordem de milhões de dólares, devido aos equipamentos caríssimos que são usados. Esse preço só é compensador quando o roteiro requer cenas impossíveis de serem obtidas na vida real.

A grande maioria das pessoas não tem acesso a computadores profissionais de animação gráfica. Portanto, devem contentar-se em usar seus próprios micros para executar tais tarefas.

A animação gráfica é uma das muitas aplicações dos programas de projeto assistido por computador (PAC), já discutido em artigos anteriores. Nesses casos, o grau de sofisticação alcançado é limitado pela capacidade do microcomputador utilizado. Um dos mais poderosos equipamentos de animação gráfica para filmes, por exemplo, é o Cray X-MP, um supercomputador que opera à velocidade de 100 megaflops (100 milhões de operações em ponto flutuante por segundo). Mas, como as imagens são muito complexas e têm que ser trocadas muitas vezes por segundo, nem mesmo o Cray consegue gerar uma animação em tempo real. Suas imagens precisam ser filmadas separadamente, quadro a quadro, e recompostas, como se faz em um desenho animado.

Uma figura sem muitos detalhes, porém, permite animações bem próximas do real. Você mesmo já deve ter visto videogames bem produzidos, em que as imagens geradas chegam à velocidade de cinqüenta quadros por segundo.

O maior problema em uma animação feita no computador é a grande quantidade de informações existentes em um desenho. Quanto mais detalhada a figura, mais memória é exigida para seu armazenamento. Igualmente, quanto mais colorida for a imagem, mais espaço de memória RAM é necessário.

Devemos também levar em conta que, à medida que cresce a quantidade de informações a serem processadas pela UCP, mais lenta se torna a animação. É por esse motivo que nem mesmo os mais sofisticados computadores são auto-suficientes na produção de um filme ou desenho animado. Não existe ainda uma UCP tão rápida a ponto de dispensar, totalmente, o processo manual de montagem quadro a quadro.

Se a animação de figuras é uma tarefa lenta para os mais avançados computadores, como você poderá movimentá-las na tela de seu micro? Uma das soluções consiste em lançar mão das páginas gráficas.

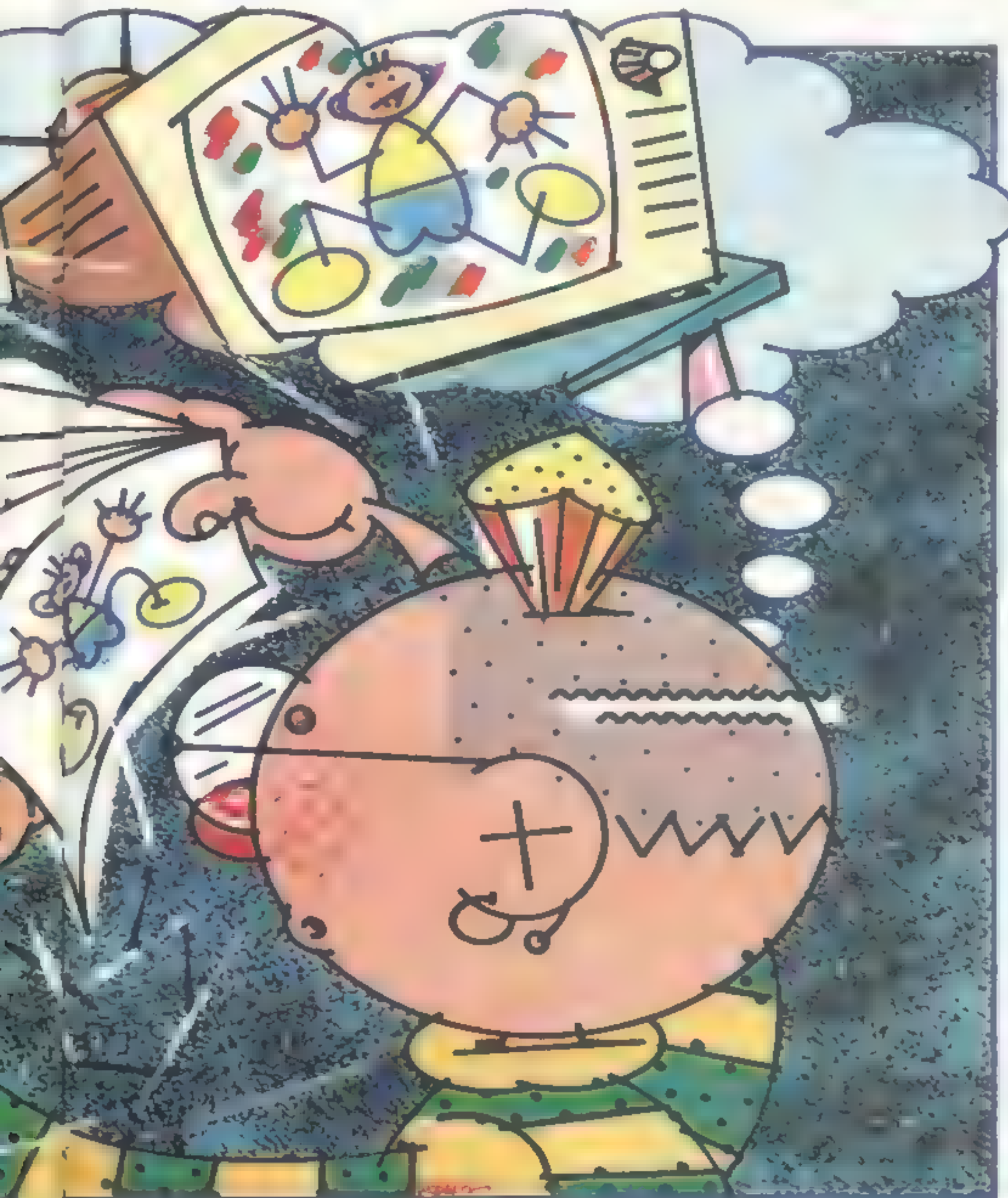


- A PERSISTÊNCIA DA VISÃO
- A LANTERNA MÁGICA
- O DESENHO ANIMADO
- GRÁFICOS NO COMPUTADOR
- PÁGINAS GRÁFICAS
- MOVIMENTAÇÃO DE CUBO
- CRIANDO SUA PRÓPRIA ANIMAÇÃO

O QUE SÃO PÁGINAS GRÁFICAS

Todos os microcomputadores possuem uma área da memória reservada para a tela de vídeo. Ela pode ser de dois tipos: a *memória mapeada*, na qual a cada ponto da tela corresponde um local na memória; ou o *arquivo de códigos*, organizado como uma lista.

No conceito das páginas gráficas, em vez de se construir a figura diretamente na memória reservada para a tela, usamos uma outra área da memória RAM, definida especialmente para esse fim. Assim que se conclui a figura, ela é transferida para a parte da RAM normalmente associada à tela. A área utilizada para construir o desenho é chamada de *página gráfica*.



Mas, qual a vantagem de empregá-la? Seguramente não haverá nenhuma economia de tempo, se tivermos que desenhar a figura primeiro em uma área separada da memória. Ao contrário, quando se transferem as informações para a memória de tela, perde-se um certo tempo. A vantagem reside no fato de que podemos alterar à vontade a página "oculta", sem provocar simultaneamente alterações na tela.

É óbvio que a técnica de páginas gráficas não tem muito valor quando se trata de desenhar apenas uma figura. Suponhamos, porém, que você queira escrever um programa em que há um texto seguido de um desenho: seria bem mais conveniente ir construindo a figura em algum lugar da memória, enquanto o usuário estivesse ocupado com a leitura do vídeo. Com a tela gráfica, muito tempo seria economizado, uma vez que ela já estaria pronta e disponível em uma parte da memória.

Mas a grande vantagem das páginas gráficas se evidencia quando se pretende mostrar uma sucessão muito rápida de figuras. Os comandos em BASIC em geral escrevem apenas nas áreas reservadas para a tela. Isso significa que iremos construir a figura nesta área, e, depois, transportá-la para outra parte da RAM. Esse processo será mais lento na etapa de montagem, mas muito mais rápido na exibição da imagem, pois o microprocessador não precisará executar uma série de comandos e funções em BASIC. Ele apenas transferirá as informações daquela parte da memória para a tela. E, se você montar várias figuras em diferentes áreas da memória, elas poderão ser chamadas rapidamente para a tela, produzindo um efeito de animação.

O ALGORITMO DE UM CUBO

Tomemos como exemplo a montagem de uma seqüência animada que represente a rotação de um cubo. Decidiu-se que quatro quadros serão suficientes

para simular uma rotação e que o cubo dará cinco voltas.

O programa poderia ser estruturado da seguinte maneira:

```
para c = 1 até 5 faça
começo
limpar a tela
construir a figura número 1
limpar a tela
construir a figura número 2
limpar a tela
construir a figura número 3
limpar a tela
construir a figura número 4
fim
```

A idéia parece simples demais: limpa-se a tela, exibe-se cada uma das figuras em seqüência e o processo é repetido até que se completem as cinco rotações. Mas esse método apresenta uma desvantagem: os cálculos para se desenhar cada figura são refeitos a cada uma das cinco repetições. Como os cálculos tomam a maior parte do tempo no processo, ocorrerá um "pulo" entre cada figura exibida, o que resultará em um fraco efeito de animação.

Observe agora este algoritmo, que ilustra o procedimento geral usado na técnica de páginas gráficas:

```
limpar a tela
construir a figura número 1
guardar a tela na página de memória 1
limpar a tela
construir a figura número 2
guardar a tela na página de memória 2
limpar a tela
construir a figura número 3
guardar a tela na página de memória 3
limpar a tela
construir a figura número 4
guardar a tela na página de memória 4
para c = 1 até 5 faça
copie os dados da página 1 para a tela
copie os dados da página 2 para a tela
copie os dados da página 3 para a tela
copie os dados da página 4 para a tela
fim
```

O programa é mais longo e exige um demorado processo de construção das quatro figuras, antes de começar a animação. Mas, uma vez armazenadas nas páginas de memória, as figuras podem ser mostradas em rápida seqüência.

Embora a construção dos desenhos seja executada em BASIC, podemos utilizar uma rotina em código de máquina para efetuar a transferência de uma figura da tela para a página gráfica e vice-versa. Ela realizará essas transferências em um piscar de olhos — o que é a essência da animação.

FAÇA SUA PRÓPRIA ANIMAÇÃO

Os próximos programas são aplicações bem simples da técnica das páginas gráficas. Os que utilizam código de máquina devem ser gravados antes da execução, para evitar que se percam, caso ocorra algum erro de digitação. Você pode usar suas próprias figuras nesses programas, colocando os comandos de desenho nas linhas adequadas.

Em um próximo artigo, examinaremos em detalhe as técnicas aqui empregadas e você verá como aproveitar ao máximo a capacidade de seu micro.



Este programa — adequado só ao Spectrum de 48 K — produzirá a animação de um acrobata pulando sobre um



O acrobata na tela do Spectrum.



TRS-Color: o asterisco que se movimenta.

trampolim. Embora a maior parte do programa seja em BASIC, há um trecho em linguagem de máquina que dará a velocidade necessária às figuras na tela.

```
10 BORDER 0: PAPER 0: INK 7:
CLS
20 CLEAR 53230
30 GOSUB 220
40 LET srce=64: LET dest=208
50 CLS
60 CIRCLE 128,168,7: PLOT 128
,161: DRAW 0,-15: DRAW -10,
-10: PLOT 128,146: DRAW 10,
-10: PLOT 118,161: DRAW 11,-5
: DRAW 10,5
70 PLOT 108,106: DRAW 40,0:
PLOT 113,106: DRAW -8,-8:
PLOT 145,106: DRAW 8,-8
80 GOSUB 270: LET dest=dest+
16
90 PRINT AT 21,2;"qualquer te
cla para comecar": PAUSE 0
100 CLS : CIRCLE 128,151,7:
PLOT 128,134: DRAW 0,-15: DRAW
-5,-16: PLOT 128,120: DRAW 5,
-17: PLOT 118,125: DRAW 10,5:
DRAW 11,-5
110 PLOT 108,106: DRAW 15,-4:
DRAW 10,0: DRAW 15,4: PLOT 113
,105: DRAW -8,-8: PLOT 144,105
: DRAW 8,-8
120 PRINT AT 6,4;"!!BOING!!"
130 GOSUB 270
140 PRINT AT 21,2;"qualquer te
cla para comecar": PAUSE 0
150 LET srce=208: LET dest=64
160 PRINT AT 17,3;"qualquer te
cla para pular": PAUSE 0
170 FOR n=0 TO 1
180 CLS
190 GOSUB 270: LET srce=srce+
16
200 NEXT n
210 GOTO 150
220 DATA 1,0,16,17,0,0,33,0,0,
237,176,201
230 FOR i=53231 TO 53231+11
240 READ byte: POKE i,byte
250 NEXT i
260 RETURN
270 POKE 53236,dest
280 POKE 53239,srce
290 RAND USR 53231
300 RETURN
```

A linha 10 seleciona as cores da tela, da borda e do desenho: negro, negro e branco, respectivamente. A 20 reserva um espaço na memória e a 30 desvia o programa para a sub-rotina entre as linhas 220 e 260, que irá montar a rotina em linguagem de máquina, nessa área. Essa sub-rotina em BASIC lê os códigos após o comando **DATA** (linha 220) e os coloca na área reservada na memória, através do comando **POKE**.

A linha 40 define duas variáveis: **srce** e **dest**. A variável **srce** corresponde ao byte alto do endereço de onde os códigos devem ser retirados e **dest**, ao byte alto do endereço onde eles vão ser guardados. Informa-se, assim, ao computador onde ler a imagem da tela e onde colocá-la na memória. Em seguida, as linhas 60 e 70 desenham a primeira das duas figuras — o acrobata no ar. A linha 80 desvia o programa para a linha 270, onde há uma rotina que coloca os valores de **dest** e **srce** no programa em código de máquina. Depois, chama essa rotina para copiar a parte da tela em que está o acrobata.

O próximo passo consiste na criação da imagem para a segunda página de memória (linhas 100 a 120). Essa imagem é armazenada pela linha 130, que desvia o programa para a linha 270. A linha 150 troca os valores de **srce** e **dest**, o que faz com que os códigos sejam transferidos da RAM para a tela. As linhas 150 a 210 formam um laço que se encarrega de alternar as duas figuras na tela. Para interromper o programa, basta acionar <**BREAK**>.

Para montar sua própria animação (de dois quadros), você precisará modificar os comandos gráficos nas linhas 60 e 70 e 100 e 110. Futuramente, você verá que é possível usar até oito páginas gráficas em seqüência.



O programa do TRS-Color é ligeiramente diferente dos destinados aos outros computadores. Ele usa três páginas gráficas — em vez de duas — das oito possíveis acessadas pelo comando **PCOPY**. Esse comando é utilizado da seguinte maneira: **PCOPY** *número da primeira página* **TO** *número da última página* — **PCOPY 1 TO 8**, por exemplo. Na verdade, a alternância de páginas é tão rápida, que se torna necessário introduzir uma pausa entre elas.

O programa mostra uma grande estrela girando continuamente.

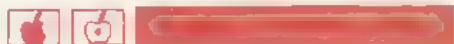
```
10 PCLEAR 8:PMODE 2,1
20 SCREEN 1,1:CLS
30 C=ATN(1)/45
50 FOR N=0 TO 2
60 PCLS
70 FOR K=0 TO 360 STEP 45
80 LINE(127,95)-(127+59*SIN(C*(
K+N*15)),95-59*COS(C*(K+N*15)))
,PSET
90 NEXT
100 PCOPY 1 TO 3+N*2:PCOPY 2 TO
 4+N*2
110 NEXT
140 FOR N=3 TO 7 STEP 2
150 PCOPY N TO 1:PCOPY N+1 TO 2
160 FOR G=1 TO 30:NEXT G,N
170 GOTO 140
```

A linha 10 abre espaço para as oito páginas gráficas e seleciona **PMODE2** na página 1, para ficar em branco e preto com média resolução. Neste modo,

uma tela ocupa duas das páginas gráficas internas. Na linha 20 o modo de alta resolução gráfica é acionado

As imagens que irão ocupar as três páginas são montadas da linha 30 até a linha 110, enquanto a linha 100 copia cada uma delas nas páginas internas. Uma tela ocupa duas páginas internas e as duas primeiras páginas são usadas para desenhar os gráficos.

As linhas 140 a 160 copiam as páginas armazenadas na tela, em seqüência. Há uma pausa na linha 160 para evitar que as imagens se alternem muito rapidamente e se sobreponham.



No Apple, contamos com duas páginas gráficas. O comando **HGR** ativa a alta resolução, limpa e mostra a página 1 da memória. O comando **HGR2** ativa a alta resolução, limpa e mostra a página 2. Colocaremos um desenho em cada página, fazendo com que se alternem. Digite e execute o programa. Você verá um beija-flor pairando no ar.

```
5   HGR
10   POKE  - 16302,0
20 X1 = 112:Y1 = 67:N = 16
30   GOSUB 100
40   FOR T = 1 TO 1000: NEXT T
50   HGR2
60 X1 = 115:Y1 = 83:N = 19
70   GOSUB 100
80   FOR T = 1 TO 1000: NEXT T
90   GOTO 300
100   FOR I = 1 TO N
110   READ X2
120   READ Y2
130   HPLOT X1,Y1 TO X2,Y2
140   LET X1 = X2: LET Y1 = Y2
150   NEXT I
160   RETURN
200   DATA     86,3,79,67,90,86,
192,3,176,64,141,99,198,166,170
,186
210   DATA  115,186,118,138,86,1
18,12,191,48,139,51,109,67,90,9
0,86
220   DATA  96,64,54,48,83,86,11
5,83,160,80,240,102,185,115
230   DATA  144,113,160,144,192,
147,179,169,144,185,115,148
240   DATA  86,118,12,191,48,139
,51,109,67,90,83,86
300   POKE  - 16304,0: POKE  - 1
6300,0
310   FOR I = 1 TO 70: NEXT I
320   POKE  - 16304,0: POKE  - 1
6299,0
330   FOR T = 1 TO 70: NEXT T
340   GOTO 300
```

A linha 5 ativa o comando **HGR**. Todos os comandos gráficos escreverão na primeira página. A linha 10 "fecha" a janela de quatro linhas na base da tela. A linha 20 define os pontos iniciais do primeiro desenho (**X1** e **Y1**) e o número de coordenadas a serem lidas (**N**). A linha 30 desvia o programa para a subrotina de desenho que vai da linha 100 à linha 160. Para construir esse primeiro desenho, serão lidas as linhas 200 e 210.

A linha 50 ativa o **HGR2**, abrindo espaço para se escrever na segunda página. A linha 60 define os valores iniciais da segunda figura e a 70 chama a subrotina de desenho, que agora irá ler as linhas 220, 230 e 240.

A rotina entre as linhas 300 e 340 alterna as duas figuras na tela. A linha 300 mostra a página 1 e a 320, a página 2. As linhas 310 e 330 introduzem pausas para diminuir a velocidade de alternância das imagens.



O programa do TK-2000 é semelhante ao do Apple. Devem ser feitas as seguintes modificacões:

```
5 MA: HOME : MP: HOME
10 MA : HGR2
50 MP : HGR2
300 MA
320 MP
```

A linha 5 limpa a primeira (**MA**) e a segunda (**MP**) páginas. A linha 10 habilita o usuário a escrever na primeira página, e a linha 50, na segunda. As linhas 300 e 320 mostram a primeira e a segunda páginas, respectivamente.

Para entender melhor o programa, veja as explicações para o Apple.



O MSX, como você já deve saber, possui uma memória exclusiva para a tela, a **VRAM**, que é dividida em oito partes de 2048 bytes. O comando **BASE** endereça cada arquivo de códigos em um deles, não necessariamente em seqüência. Como o modo de baixa resolução gráfica não exige muitas informações, grande parte da **VRAM** fica desocupada. Nesses espaços é possível criar até seis páginas gráficas. Para demonstrar isso, o próximo programa irá simular a rotação de um cubo.

```
5 BASE(18)=BASE(19)
10 BASE(17)=0:SCREEN 3
20 X1=127:Y1=40:N=4
30 GOSUB 200
40 BASE(17)=4096:SCREEN 3
50 X1=103:Y1=151:N=10
60 GOSUB 200
70 BASE(17)=6144:SCREEN 3
80 X1=95:Y1=151:N=10
90 GOSUB 200
100 BASE(17)=8192:SCREEN 3
110 X1=85:Y1=96:N=6
120 GOSUB 200
130 BASE(17)=10240:SCREEN 3
140 X1=155:Y1=151:N=10
150 GOSUB 200
160 BASE(17)=12288:SCREEN 3
170 X1=151:Y1=151:N=10
180 GOSUB 200
190 GOTO 400
200 FOR I=1 TO N
210 READ X2
220 READ Y2
230 LINE(X1,Y1)-(X2,Y2)
240 LET X1=X2:LET Y1=Y2
250 NEXT I
260 RETURN
300 DATA 177,96,127,151,74,96,1
27,40
310 DATA 133,96,176,96,151,151,
103,151,78,96,103,40,151,40,181
,96,128,96,103,40
320 DATA 110,96,170,96,155,151,
95,151,80,96,95,40,155,40,170,9
6,110,96,95,40
330 DATA 85,41,169,41,169,151,8
5,151,85,96,169,96
340 DATA 140,96,80,96,95,151,15
5,151,170,96,155,40,95,40,80,96
,140,96,155,40
350 DATA 121,96,73,96,103,151,1
51,151,181,96,151,40,103,40,73,
96,121,96,151,40
400 BASE(17)=0
410 FOR I=1 TO 70:NEXT I
420 BASE(17)=4096
430 FOR I=1 TO 70:NEXT I
440 BASE(17)=6144
450 FOR I=1 TO 70:NEXT I
460 BASE(17)=8192
470 FOR I=1 TO 70:NEXT I
480 BASE(17)=10240
490 FOR I=1 TO 70:NEXT I
500 BASE(17)=12288
510 FOR I=1 TO 70:NEXT I
520 GOTO 400
```

Primeiro, a linha 5 coloca o endereço de um arquivo que não iremos usar no último segmento da **VRAM**, junto com um outro, a fim de criar espaço. A linha 10 faz o computador colocar a tabela de padrões (a figura propriamente dita) a partir do endereço 0. A 20 define os pontos iniciais do desenho (**X1** e **Y1**) e o número de coordenadas que serão lidas (**N**). A 30 chama a rotina de desenho, que começa na linha 200 e termina na 260. A 300 contém as informações para o primeiro desenho.

Feitos os desenhos, o programa é desviado para a rotina da linha 400, que alterna as figuras na tela, dizendo ao computador, por meio do comando **BASE**, onde buscá-las. Essa rotina inclui ainda uma pausa entre as figuras, para que elas não se sobreponham.

ARMAZENAGEM DE PROGRAMAS

O funcionamento do computador desperta nossa curiosidade sobretudo quando algo não vai bem. Sabendo como os programas são armazenados, você poderá detectar seus erros com mais facilidade.

O que o computador faz com seu programa depois de tê-lo digitado?

Provavelmente, você sabe que ele é armazenado em um lugar especial da memória, reservado para programas em BASIC. Tem, porém, idéia de onde fica esse lugar e de que maneira as linhas são escritas, para que o computador as entenda?

Na memória do computador, o programa se assemelha bastante à listagem que você digitou, e muitos dos números ali guardados são códigos ASCII, representando as letras. Mas a máquina re-



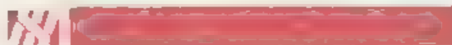
duz algumas palavras — as palavras-chave — e, também, introduz informações extras entre as linhas, para facilitar sua localização.

O computador possui ainda uma área onde guarda as variáveis criadas pelo programa, modificando-as sempre que necessário. Quando digitamos, por exemplo, **10 LET A = 6**, a máquina não só armazena essa linha como coloca o nome e o valor da variável nela contida em outro local da memória.

Vejamos tudo isso um pouco mais de perto. Para começar, digite o próximo programa exatamente como está. No Spectrum, não deixe nenhum espaço. Nos outros micros, coloque um espaço apenas após as palavras PRINT e LET.



```
10 PRINT "INPUT"
20 LET A=2
30 PRINT A
40 STOP
```



A armazenagem de um programa em BASIC, no MSX, começa no endereço 32769. Para ler os códigos armazenados, a partir desse endereço, dê o seguinte comando direto:

```
FOR I=32769 TO 32808:PRINT PEEK
(I);" ";:NEXT I
```

Obteremos, então, este bloco:

15	128	10	0	145
32	34	73	78	80
85	84	34	0	25
128	20	0	136	32
65	239	19	0	33
128	30	0	145	32
65	0	39	128	40
0	144	0	0	0

Se você considerar esses números como códigos ASCII e tentar relacioná-los aos caracteres equivalentes, descobrirá que só alguns fazem sentido:

?	?	?	?	?
esp	"	I	N	P
U	T	"	?	?
?	?	?	?	esp
A	?	?	?	?
?	?	?	?	esp
A	?	?	?	?
?	?	?	?	?

Já se vê parte do programa. Mas onde estão o **PRINT**, que antecede a palavra INPUT, e os demais comandos?

O MSX usa um artifício para economizar memória: codifica as palavras e os caracteres reservados, substituindo-os, ao armazenar o programa, por esses códigos, usualmente chamados *tokens* (do inglês: símbolo, indicação).

Se, posteriormente, você tiver interesse em obter uma lista dos tokens, utilize este programa:

```
10 E=14962
20 C=65
30 PRINT CHR$(C);
40 A=PEEK(E)
50 B=PEEK(E+1)
60 A$=CHR$(A)
70 IF A<128 THEN PRINT A$,:GOTO
 110
80 PRINT CHR$(A-128);TAB(8);B
90 E=E+1
100 IF PEEK(E+1)<>0 THEN PRINTC
HR$(C);
110 IF PEEK(E+1)<>0 THEN 150
120 C=C+1:IF C=89 THEN 170
125 B$=CHR$(C):PRINT
130 IF B$="J" OR B$="Q" THEN 15
0
140 PRINT B$;
150 E=E+1
160 IF E<=15649 THEN 40
170 E=15654:PRINT
180 A=PEEK(E):B=PEEK(E+1)
190 IF A>127 THEN PRINT CHR$(A-
128);
200 IF A<128 THEN PRINT CHR$(A)
;
210 PRINT TAB(8);B
220 E=E+2:PRINT
230 IF E<15673 THEN GOTO 180
```

Interpretando os tokens, nosso bloco ficará assim:

?	?	?	?	PRINT
esp	"	I	N	P
U	T	"	?	?
?	?	?	LET	esp
A	=	?	?	?
?	?	?	PRINT	esp
A	?	?	?	?
?	STOP	?	?	?

Na terceira, 17ª, 27ª e 35ª posições encontra-se a seqüência 10, 20, 30 e 40. Esses bytes guardam a parte baixa (L) do número de cada linha e cada byte subseqüente guarda a parte alta (H). Os dois bytes anteriores armazenam, do mesmo modo, o endereço da próxima linha.

120L	120H	10	0	PRINT
esp	"	I	N	P
U	T	"	?	130L
130H	20	0	LET	esp
A	=	?	?	140L
140H	30	0	PRINT	esp
A	?	fimL	fimH	40
0	STOP	?	?	?

Para os bytes finais, o valor 0 significa fim de linha; dois 0 seguidos indicam fim de programa. O valor 2, atribuído à variável **A**, é armazenado no 23º byte, de uma forma relativamente complicada, que não cabe aqui examinar. Finalmente, temos o diagrama completo de uma linha na RAM:

120L	120H	10	0	PRINT
esp	"	I	N	P
U	T	"	fiml	130L
130H	20	0	LET	esp
A	=	2	fiml	140L
140H	30	0	PRINT	esp
A	fiml	fimL	fimH	40
0	STOP	fiml	fimpr	fimpr

VARIÁVEIS NUMÉRICAS

Já sabemos como um programa é colocado na memória do micro. Veremos agora como o valor de uma variável é armazenado na RAM. Lembre-se de que o MSX possui a função **VARPTR** (do inglês **VAR**iable **P**oin**T**e**R**), que mostra o endereço a partir do qual está guardada determinada variável.

Vamos explorar, em primeiro lugar, as variáveis numéricas.

Use **NEW** para apagar o programa anterior e execute este.

```
10 DEFINT A
20 A=1
30 V=VARPTR(A)-2
40 FOR K=V TO V+3
50 PRINT PEEK(K),
60 NEXT
```

Você obterá os seguintes números:

65 0 1 0

O primeiro e o segundo bytes armazenam o nome da variável — por isso, só as duas primeiras letras dela importam. Como se trata de uma variável do tipo inteira, bastam os dois bytes seguintes para guardar seu valor nas duas diferentes modalidades (parte baixa, L, e parte alta, H).

Faça as seguintes substituições no programa e volte a executá-lo:

```
10 DEFSNG A
40 FOR K=V TO V+6
```

Temos então estes números:

65 0 65 16 0 0

Os dois primeiros bytes dão o nome à variável. Estamos lidando agora com uma variável de precisão simples e, por isso, precisaremos de cinco bytes para armazenar seu valor. O primeiro deles cuida do expoente, guardando o resultado da soma de seu valor com o número 65, para facilitar o manejo de expoentes negativos. Os três bytes restantes encarregam-se de guardar o número propriamente dito.

Faça estas novas substituições e execute o programa.



```
10 DEFDBL A
4. FOR K=V TO V+9
```

Aparecem na tela os números:

```
65  0  65  16  0  0  0  0  0  0
```

Como você pode observar, o nome e o expoente continuam sendo armazenados da mesma maneira, mas são necessários sete bytes para guardar o número.

VARIÁVEIS STRING

Para armazenar as variáveis do tipo *string*, o MSX utiliza três bytes. Digite o próximo programa e veja como eles são aproveitados.

```
1  A$="AB"
2  V=VARPTR(A$)-2
3C FOR K=V TO V+4
40 PRINT PEEK(K);" ";
5. NEXT
```

Você deve ter obtido os números:

```
65  0  3  9  128
```

Os dois primeiros bytes contêm o nome da variável; o terceiro, seu comprimento; os dois últimos indicam o endereço no qual ela se encontra, ou seja, sua posição exata na parte da RAM que armazena o programa. Este é um artifício que o MSX utiliza para economizar memória. Se houver alguma modificação na linha que contém a variável (a linha 10), que impeça o computador de usá-la, ele colocará seu conteúdo em outro endereço.

VARIÁVEIS INDEXADAS

Quando você define uma matriz com o comando **DIM(n)**, o MSX reserva, na memória, um espaço para **n**+1 variáveis, pois a primeira delas tem índice 0. Execute o próximo programa e veja como elas são armazenadas.

```
10 DEFINT A
2) .IM A 2
3( A ) 5 A(. = 0 A(2)=15
40 V VARPTR.A 0))
50 FOR K V T( V+16
60 PRINT PEEK(K,.
70 NEXT K
```

Você deve ter obtido:

```
0  0  0  2  65  0  9  0  1  3
0  5  0  10  0  15  0
```

Os três primeiros bytes zerados inicializam a área de armazenamento. O byte com o número 2 indica que a variável é do tipo inteira — seria 4, para variáveis de precisão simples, e 8, para variáveis de precisão dupla. Os dois bytes seguintes fornecem o nome da variável. O sétimo e o oitavo indicam quantas posições faltam para o fim dessa área. O nono byte contém o número de dimensões, e o décimo e o 11º, o tamanho dessas dimensões. Em seguida vêm os números, armazenados conforme o tipo de cada variável.

Para as variáveis indexadas do tipo string pouca coisa mudaria. O quarto byte, que indica o tipo de variável, teria o número 3, e, em vez de guardar um valor numérico, conteria os endereços dos caracteres, como no armazenamento de strings comuns.

No Spectrum, há dois endereços de memória que, juntos, nos dão o endereço a partir do qual está armazenado o programa em BASIC. Acesse-os com o seguinte comando direto:

```
PRINT PEEK 23635+256*PEEK 23636
```

Você obterá como resposta o número 23755, em um Spectrum de 48K. Se você achou um número diferente e tem certeza de que digitou o comando corretamente, substitua 23755 pelo valor encontrado, no próximo comando.

```
FOR I=23755 TO 23755+40:PRINT
PEEK I;" ";:NEXT
```

Você obterá os seguintes números

```
0    10   9    0    245
34   73   78   80   85
84   34   13   0    20
11   0    241  65   61
50   14   0    0    2
0    0    13   0    30
3    0    245  65   13
0    40   2    0    226
13
```

Agora, usando os códigos ASCII, observe como fica o quadro:

```
?    ?    ?    ?    ?
"    I    N    P    U
T    "    ?    ?    ?
?    ?    ?    A    =
2    ?    ?    ?    ?
?    ?    ?    ?    ?
?    ?    ?    A    ?
?    ?    ?    ?    ?
?
```

Você já pode ver parte do programa, mas falta decifrar vários códigos.

Provavelmente o comando **PRINT** estaria antes de "INPUT", mas o que vemos nessa posição é o número 245. O Spectrum usa valores acima de 164 para codificar as palavras reservadas. Se você consultar a lista delas, no manual, verá que 245 é o valor, chamado *token* (do inglês: símbolo, indicação), para **PRINT**. Consultando a lista de tokens, localizaremos os comandos.

```
?    ?    ?      ?      PRINT
"    I    N      P      U
T    "    ?      ?      ?
?    ?    LET    A      =
2    ?    ?      ?      ?
?    ?    ?      ?      ?
?    ?    PRINT  A      ?
?    ?    ?      ?      STOP
?
```

No segundo, 15°, 30º e 37º bytes, encontra-se a seqüência 10, 20, 30, 40. Esses bytes são usados para guardar a parte baixa (L) do número da linha, assim como o byte anterior guarda a parte alta (H). Os dois bytes seguintes da linha indicam seu tamanho (excluídos os quatro primeiros), sendo que, agora, a parte baixa aparece primeiro.

Temos, assim, o seguinte diagrama:

```
0      10     compL  compH  PRINT
"      I      N      P      U
T      "      ?      0      20
compL  compH  LET    A      =
2      ?      ?      ?      ?
?      ?      ?      0      30
compL  compH  PRINT  A      ?
0      40     compL  compH  STOP
?
```

Vejamos, agora, os bytes finais. O valor **13** é usado para indicar fim de linha (fdl). O número **14** indica que os próximos quatro bytes são uma forma codificada de uma constante numérica — no caso, 2.

```
0      10     compL  compH  PRINT
"      I      N      P      U
T      "      fdl    0      20
compL  compH  LET    A      =
2      cnum   0      0      2
0      0      fdl    0      30
compL  compH  PRINT  A      fdl
0      40     compL  compH  STOP
fdl
```

ARMAZENAGEM DAS VARIÁVEIS

Vimos como um programa em BASIC é armazenado. Todas as variáveis definidas por ele, porém, são guardadas em uma área separada. Dois endereços — 23627 e 23628 — contêm o início dessa área, e dois outros — 23641 e 23642 — apontam para o fim. Para imprimir todo o conteúdo da memória, entre este programa:

```
10 FOR A=PEEK 23627+256*PEEK
23628 TO PEEK 23641+256*PEEK
```

```
23642
20 PRINT PEEK A;"  ";
30 NEXT A
```

VARIÁVEIS NUMÉRICAS

Em primeiro lugar, vamos definir uma variável, acrescentando esta linha ao programa anterior.

```
1 LET B=150000
```

Executando o programa, temos:

```
98   146   18   124   0   0   225   0
B    ←———————— valor ————————→
```

O número 98 é o código para a letra **B** mais 32 — isto mostra ao computador que ele está lidando com uma variável numérica. O valor 150000 é armazenado nos cinco bytes seguintes, na forma de ponto flutuante. Os dois últimos bytes, 225 e 0, indicam o fim da área de variáveis. Como eles sempre aparecerão na mesma posição, deixaremos de mencioná-los nos próximos exemplos.

As variáveis com nomes longos são armazenadas da mesma maneira, apesar dos caracteres terem seus códigos um pouco alterados. Tente isto:

```
1 LET MARIA=30
```

Você deve ter obtido os números

```
173   97   114   105   225
 M     A    R     I     A

0   0   30   0   0
←——— valor ———→
```

Os primeiros cinco bytes guardam o nome MARIA. À primeira letra soma-se o número 96, indicando um longo nome; às do meio, o número 32; à ultima letra é adicionado 160, indicando o fim do nome. Os últimos cinco bytes armazenam o valor numerico.

VARIÁVEIS STRING E INDEXADAS

A próxima linha de programa mostra a maneira como as variáveis do tipo *string* são armazenadas:

```
1 LET A$="STRING"
```

Você deve ter obtido:

```
65  6  0  83  84  82  73  78  71
A          S   T   R   I   N   G
```

O nome da variável, **A**, vem seguido de dois bytes com a quantidade de caracteres armazenados. Como você pode notar, estes caracteres são representados pelos códigos em ASCII.

Tente agora uma variável numérica indexada:

```
1 DIM F(2)
2 LET F(1)=100
3 LET F(2)=200
```

Executando o programa, temos:

```
134  13  0  1  2  0  0  0  100  0  0
 F                  ←—— valor ——→

0  0.200  0  0
←— valor —→
```

O primeiro byte é **F** mais 64, indicando uma variável numérica indexada. Os dois próximos bytes, com a parte baixa primeiro, indicam o número de bytes seguintes — 13. O próximo byte diz o número de dimensões e os dois seguintes mostram o número de elementos reservados. Só então se chega aos dois valores armazenados, que ocupam cinco bytes cada um.

Veja, finalmente, as variáveis indexadas do tipo string. Digite estas linhas e execute o programa.

```
1 DIM B$(2,8)
2 LET B$(1)="NOVA"
3 LET B$(2)="CULTURAL"
```

Observe como fica o quadro:

```
194  21  0  2  2  0  8  0
 B

78  79  86  65  32  32  32  32
N   O   V   A

67  85  76  84  85  82  65  76
C   U   L   T   U   R   A   L
```

O primeiro byte e o codigo de **B** mais 128, para variáveis indexadas do tipo string. Os dois próximos mostram quantos bytes ainda serão encontrados até o fim da área de variáveis. O byte seguinte guarda o número de dimensões — neste caso, 2. Outros dois pares de bytes contêm, respectivamente, o número de elementos e o comprimento máximo deles. Os bytes restantes guardam os caracteres, sendo que o codigo 32 (espaço) completa o espaço reservado para um elemento, se este for menor que o comprimento reservado.



No TRS-Color, existem dois endereços que, juntos, indicam o início da área de programas em BASIC. Esses endereços são 25 e 26 (decimal).

Digite o seguinte comando:

```
PRINT PEEK(25)*256+PEEK(26)
```

Se seu computador acabou de ser ligado, a resposta será 7681. Caso obtenha outro número, substitua-o no seguinte comando que irá executar:

```
FOR I=7681 TO 7681+39:PRINT
PEEK(I);:NEXT I
```

Você deve ter obtido este bloco:

30	15	0	10	135
32	34	73	78	80
85	84	34	0	30
25	0	20	142	32
65	203	50	0	30
33	0	30	135	32
65	0	30	39	0
40	146	0	0	0

Agora, usando os códigos ASCII, observe como fica o quadro:

?	?	?	?	?
esp	"	I	N	P
U	T	"	?	?
?	?	?	?	esp
A	?	2	?	?
?	?	?	?	esp
A	?	?	?	?
?	?	?	?	?

Já podemos ver uma parte do programa em BASIC, mas existem ainda muitos pontos de interrogação.

Você poderia esperar que a palavra **PRINT** aparecesse antes de **"INPUT"**, no diagrama. Em vez disso, temos o número 135. O computador usa valores acima de 127 para representar as palavras reservadas. O numero 135 e o valor, também chamado *token* (do inglês: símbolo, indicação), para o comando PRINT. Com esses valores, você pode tornar o diagrama mais completo.

?	?	?	?	PRINT
esp	"	I	N	P
U	T	"	?	?
?	?	?	LET	esp
A	=	2	?	?
?	?	?	PRINT	esp
A	?	?	?	?
?	STOP	?	?	?

Se quiser, depois, obter a lista dos tokens, use este programa:

```
10 CLEAR 1000:CLS
20 C=43622:FOR K=0 TO 78
30 W$(K/39)=W$(K/39)+CHR$(PEEK(
C))
40 C=C+1 IF PEEK(C 1)<128 THEN
30
45 IF K=52 THEN C=33155
50 NEXT
60 C=43802 FOR K=1 TO 34
70 F$=F$+CHR$(PEEK(C))
80 C=C+1:IF PEEK(C 1)<128 THEN
70
85 IF K=20 THEN C=33310
90 NEXT
```

```
100 PRINT·INPUT" DIGITE O SIMBO
LO EM HEXADECIMAL";TS
110 TK=VAL("&H"+TS)
120 IF TK>65280 THEN TK=TK-6528
0·GOTO 210
130 IF TK<128 OR TK>205 THEN 25
0
140 K=-39*(TK>166):P=1
150 IF K=TK-128 THEN WS=W$(K/39
)·GOTO 180
160 P=P+1:IF ASC(MID$(W$(K/39),
P-1,1))<128 THEN 160
170 K=K+1·GOTO 150
180 PRINT:PRINT " SIMBOLO ";TS;
" = ";
190 AS=MIDS(WS.P.1).IF AS<CHRS(
128) THEN PRINT AS;:P=P+1:GOTO
190
200 PRINT CHRS(ASC(AS) AND 127)
:GOTO 100
210 K=0 P=1.IF TK<128 OR TK>161
 THEN 250
220 IF K=TK-128 THEN WS=FS:GOTO
 180
230 P=P+1:IF ASC(MID$(F$,P-1,1)
)<128 THEN 230
240 K=K+1·GOTO 220
250 PRINT" CARACTER ILEGAL ":PR·
INT:GOTO 100
```

Digitando qualquer token, o programa fornecerá a função ou comando equivalente. Seus valores, em hexadecimal, vão de 80 até CD, para os comandos, e de FF80 até FFA1 para as funções.

No quarto, 18.º, 28.º e 36.º bytes há esta sequência: 10, 20, 30, 40. Esses bytes são usados para armazenar a parte baixa (L) do número da linha, enquanto o zero antes deles guarda a parte alta (H).

Os dois primeiros bytes de cada linha são usados para indicar onde começa a linha seguinte. Eles podem ser, por exemplo, 30 e 15. Multiplicando 30 por 256 — pois se trata da parte alta — e adicionando 15, temos 7695, número correspondente ao endereço que contém o primeiro byte da linha 20.

Finalmente, sabendo que o valor 0 significa fim de linha, podemos completar o diagrama:

prxlH	prxlL	0	10	PRINT
esp	"	I	N	P
U	T	"	fdl	prxlH
prxlL	0	20	LET	esp
A	=	2	fdl	prxlH
prxlL	0	30	PRINT	esp
A	fdl	prxlH	prxlL	0
40	STOP	fdl	0	0

ARMAZENAGEM DAS VARIÁVEIS

Vimos como um programa é armazenado. Porém, sempre que se define uma variável no programa, ela é colocada separadamente em duas áreas especiais. Há dois endereços, 27 e 28, que indicam o início da área utilizada para guardar variáveis simples. Outros dois endereços, 29 e 30, apontam para o início da área de variáveis indexadas.

As variáveis simples também podem ser acessadas por meio do comando **VARPTR**, seguido do nome da variável. Então, **VARPTR(A)** fornecerá a localização do valor da variável **A**. Como o nome dessa variável é armazenado imediatamente antes de seu valor, usaremos **VARPTR(A) – 2**.

VARIÁVEIS SIMPLES

O próximo programa imprime os seis primeiros bytes da área de variáveis simples, mostrando como a variável **A** é armazenada na memória.

```
1 A=1
10 V=VARPTR(A)-2
20 FOR K=V TO V+6
30 PRINT PEEK(K);
40 NEXT:PRINT
```

Você obterá os seguintes números:

```
65 0 129 0 0 0 0
A     ←— valor    →
```

A letra **A** é armazenada primeiro e o byte seguinte deixa um espaço reservado para outra letra. Os cinco bytes restantes são o número 1, na forma de ponto flutuante.

VARIÁVEIS DO TIPO STRING

As variáveis *string* são armazenadas de forma muito semelhante. Troque a letra **A** da linha 10 por **AA$** e substitua a linha 1 por:

```
1 AAS="CADEIRA"
```

Desta vez, você terá:

```
65 193 7 0 38 10 0
 A   A
```

O nome **AA** é colocado nos dois primeiros bytes. Adiciona-se à segunda letra o valor 128, para indicar que ela é uma variável string. O número 7 corresponde à quantidade de bytes usados (conte-os). O 38 e o 10 são, respectivamente, a parte alta e a parte baixa do endereço onde a palavra **CADEIRA** está armazenada.

Calculando esse endereço, temos 38*256+10, que é igual a 9738. Efetivamente, esta é a localização da palavra CADEIRA no programa em BASIC. O TRS-Color, desse modo, economiza memória, evitando que a informação seja duplicada. Mas, assim que se alterar a linha ou a variável, esta será colocada em um outro lugar, e o endereço calculado não será mais o mesmo. Os zeros que cercam os dois números mostram sua utilidade quando o computador apaga velhas variáveis e cria outras.

VARIÁVEIS INDEXADAS

O próximo programa ajudará a vasculhar a área de variáveis indexadas.

```
10 T=1:K=1:V=PEEK(29)*256+PEEK(
30)
20 T=PEEK(V+2)*256+PEEK(V+3)
30 FOR K=V TO V+T-1
40 PRINT PEEK(K);
50 NEXT
```

Acrescente estas linhas e depois execute o programa.

```
1 DIM A(0,2)
2 A(0,0)=10
3 A(0,2)=20
```

Você obterá os seguintes números:

```
65 0 0 24 2 0 3 0 1
A

132 32 0 0 0 0 0 0 0 0
←— valor     → ← valor  →

133 32 0 0 0
←— valor     →
```

Como sempre, os dois primeiros bytes dão o nome da variável, e os dois outros indicam o seu comprimento. O byte seguinte mostra o número de dimensões, 2, e os dois próximos pares, o número de elementos, na ordem inversa — o 03 e o 01 indicam uma matriz 1 por 3. Depois disso, vêm os próprios valores, na forma de ponto flutuante. Observe que foi reservado um espaço para o elemento **(0,1)**, embora ele não tenha sido definido.

Faça uma experiência com as variáveis string indexadas.

```
1 DIM A$(2)
2 AS(0)="NOVA"
3 AS(2)="CULTURAL"
```

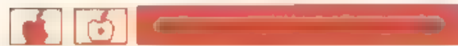
Executando o programa, obterá:

```
65 128 0 22 1 0 3 4 0 38 24 0 0

0 0 0 0 8 0 33 41 0
```

O número 65 corresponde à letra **A** e o 128 indica uma variável string. Há dois bytes para o comprimento, um para a dimensão e um par destinado à quantidade de elementos. Em seqüência, encontram-se três grupos de cinco bytes por cadeia. Cada grupo começa por um

número que diz quantos caracteres a variável possui — 4, no caso de **NOVA**. Seguem-se um zero para guardar uma casa, dois bytes para o endereço da variável e mais um zero. O mesmo se repete para todas as variáveis, inclusive as que não foram definidas. Os valores 38 e 33 poderão ser diferentes se se colocar o programa em uma parte diferente da memória.



Tanto o Apple como o TK-2000 começam a armazenar os programas em BASIC a partir do endereço 2048. Depois de digitar o programa inicial, execute o seguinte comando para ver como ele foi colocado na memória:

```
FOR I=2048 TO 2083:PRINT PEEK(I
);" ";:NEXT
```

Você obterá estes números:

```
0     14    8     10    0
186   34    73    78    80
85    84    34    0     23
8     20    0     170   65
208   50    0     30    8
30    0     186   65    0
36    8     40    0     179
0
```

Vejamos se algo faz sentido, se usarmos os códigos ASCII:

```
?     ?     ?     ?     ?
?     "     I     N     P
U     T     "     ?     ?
?     ?     ?     ?     A
?     2     ?     ?     ?
?     ?     ?     A     ?
?     ?     ?     ?     ?
?
```

Talvez você esperasse que a palavra **PRINT** aparecesse antes da palavra INPUT. Porém, a fim de economizar memória, o computador substitui as palavras e os caracteres reservados por códigos, denominados *tokens* (do inglês: símbolo, indicação). Consulte a lista de tokens no manual de seu micro e complete o diagrama.

```
?     ?     ?     ?     ?
PRINT "     I     N     P
U     T     "     ?     ?
?     ?     ?     LET   A
=     2     ?     ?     ?
?     ?     PRINT A     ?
?     ?     ?     ?     STOP
?
```

Já podemos ver uma parte do programa, mas ainda há muitos pontos de interrogação. Se você der uma olhada no quarto, 17º, 26º e 33º bytes, encontrará uma sequência de números — 10, 20, 30, 40. Esses bytes guardam a parte baixa (L) do número de cada linha, e o byte seguinte guarda a parte alta (H). Os dois bytes anteriores contêm o endereço onde se inicia a próxima linha. O valor zero serve para separar as linhas de programa.

```
inic  120H  120L  10    0
PRINT "     I     N     P
U     T     "     fdl   130H
130L  20    0     LET   A
=     2     fdl   140H  140L
30    0     PRINT A     fdl
fimH  fimL  40    0     STOP
fim
```

ARMAZENAGEM DAS VARIÁVEIS

Já vimos como os programas são armazenados. Porém, sempre que se define uma variável, ela é guardada em uma parte especial da memória. O endereço da área de variáveis simples é dado pelos bytes 105 (L) e 106 (H).

Digite este programa e execute-o para ver como elas são armazenadas.

```
1  LET A=3
10 V=PEEK(105)+256*PEEK(106)
20 FOR I=V TO V+6
30 PRINT PEEK(I);
40 NEXT I
```

Teremos, então:

```
65 0 130 64 0 0 0
```

O número 65 é o código ASCII para a letra A, que corresponde ao nome da variável. O byte seguinte tem a mesma finalidade. O terceiro código refere-se ao expoente, e os quatro restantes, finalmente, guardam o valor da variável A armazenagem dos números, um tanto complicada, não interessa diretamente aos objetivos deste artigo, e, portanto, não será aqui examinada.

Para uma variável do tipo *string*, faça a seguinte substituição:

```
1 LET AB$="APPLE"
```

Executando o programa, você obterá:

```
65 194 5 11 8 0 0
```

Os dois primeiros bytes dão o nome da variável, mas, agora, o valor 128 é adicionado à segunda letra, para indicar uma variável string. O terceiro byte guarda seu comprimento. O par de bytes seguintes contém o endereço a partir do qual ela está armazenada. Para economizar memória, em vez de escrever novamente a palavra **APPLE** em outro lugar da memória, o computador guarda apenas a posição que ela ocupa no programa em BASIC: as variáveis string, definidas por um comando **INPUT**, por exemplo, são guardadas em uma outra posição.

VARIÁVEIS INDEXADAS

As variáveis indexadas são armazenadas em um outro endereço, dado pelos bytes 107 e 108. O computador também guarda aí algumas variáveis especiais, como as usadas nos laços **FOR...NEXT**. Por isso, pulamos sete bytes para chegar à variável **A**.

Execute o programa:

```
1 DIM A(2)
2 A(0)=5 A(1)=10 A(2)=15
10 V=PEEK(107)+256*PEEK(108)
20 FOR I=V+7 TO V+28
30 PRINT PEEK(I);" ";
40 NEXT I
```

Você deve ter obtido a sequência

```
65 0 22 0 1 0 3 131 32 0 0 0

132 32 0 0 0 132 112 0 0 0
```

Os dois primeiros bytes dão o nome da variável. O próximo par diz, na forma LH, quantos bytes foram necessários para o armazenamento (conte-os). O quinto deles guarda o número de dimensões. A partir deste, na ordem decrescente, seriam escritos os tamanhos de cada dimensão; como nós só temos uma, bastam os valores 0 e 3 (note que agora a parte alta veio em primeiro lugar). Finalmente, aparecem os valores, cada um ocupando cinco bytes, como já comentamos.

Para examinar a armazenagem das variáveis string indexadas, substitua 28 por 22 na linha 20 e mude as linhas:

```
1 DIM B$(2)
2 B$(1)="NOVA"
3 B$(2)="CULTURAL"
```

Você deve ter tido esta resposta

```
66  128  16  0  1  0  3  0  0

0  4  23  8  8  40  8
```

Os dois primeiros bytes guardam o nome, como para as variáveis string comuns. O par seguinte mostra o espaço ocupado por essa variável (novamente, conte os bytes). O quinto byte guarda o número de dimensões e os valores 0 e 3 indicam, na forma HL, o comprimento da n-ésima dimensão (no nosso caso, a única). A partir daí, cada string ocupa três bytes, sendo que o primeiro contém seu comprimento e os dois outros, sua posição no programa em BASIC. Os três zeros referem-se à variável **B$(0)**, que nós não definimos.

UMA PLANILHA ELETRÔNICA (1)

Se você vive às voltas com uma grande quantidade de informações numéricas, peça socorro ao seu micro — ele, sem dúvida, se entende melhor com números e contas do que você.

Um problema que aflige a maioria das pessoas é o controle de suas despesas. No artigo da página 134, apresentamos um programa bem prático para a organização do orçamento. Veremos, agora, um método desenvolvido a partir da planilha de cálculo utilizada pelos profissionais em contabilidade.

O programa para a planilha eletrônica é extremamente versátil, tendo um potencial quase ilimitado para a manipulação de informações numéricas. E suas aplicações não se restringem à área financeira. Para começar, examinaremos todas as suas possibilidades. Assim, depois de digitar o programa — que será dado em três partes —, você estará apto a usá-lo eficientemente. Instruções detalhadas serão fornecidas junto com as listagens.

O QUE É UMA PLANILHA DE CÁLCULO?

As planilhas de cálculo eletrônicas valem-se de uma das maiores vantagens do computador — sua capacidade de fazer cálculos muito rapidamente. Em essência, mesmo o maior e mais complexo dos computadores é uma máquina de somar. Os computadores, na verdade, só sabem lidar com números — e quem já se aventurou a trabalhar com código de máquina pode confirmar isto.

Bem utilizada, uma planilha eletrônica torna-se uma ferramenta bastante poderosa. Sua aplicação mais frequente é na área financeira, mas é possível estendê-la a qualquer finalidade. Ela substitui o papel, o lápis e a calculadora, antes usados pelos contadores para prever os lucros de uma empresa ou por cientistas investigando o crescimento populacional. No plano doméstico, pode ser muito útil no controle das despesas ou na organização dos dados relacionados a um hobby.

A planilha tradicional dos contadores, empregada para anotar os lucros e despesas, é uma folha de papel dividida horizontalmente em linhas e verticalmente em colunas. No cabeçalho, em geral, anotam-se os meses do ano, de maneira que cada coluna corresponde a um mês. Ao lado, colocam-se rótulos como receitas e despesas. Para análises mais detalhadas incluem-se subtítulos, como vendas, exportações, custos operacionais, custos de matéria-prima etc. Cada linha corresponde a um item específico de receita ou despesa.

O título final da página é, usualmente, *Lucros/Perdas* e os números das colunas mostram quanto se ganhou ou se perdeu a cada mês. Na 13ª coluna, registra-se o total da receita ou despesa, no ano, por área específica.

O preenchimento das *células* com os números é sempre uma tarefa trabalhosa, quer se recorra ou não ao auxílio de um computador. Mas os contadores que usam planilhas tradicionais vêem-se obrigados, além disso, a fazer as contas dos totais, o que significa somar todos os valores de receita, todos os valores de despesa e, finalmente, calcular a diferença.

O COMPUTADOR ENTRA EM JOGO

Em muitos aspectos, a planilha de cálculo eletrônica é igual à que foi descrita até o momento, dividindo-se também em células formadas por linhas e colunas. Para se obter células de tamanho aceitável, apenas uma pequena parte da planilha é exibida na tela, que é usada como uma "janela" para mostrar uma área particular.

Como na planilha de papel, pode-se colocar qualquer tipo de dado nas células vazias. Elas só adquirem um significado particular depois de definidas, podendo conter títulos, rótulos, valores etc., segundo a necessidade.

Até aqui, a planilha eletrônica é tão ou mais trabalhosa que a manual. Sua grande vantagem reside, de fato, na manipulação dos dados. Atrás de suas células em branco encontra-se uma outra planilha, que diz ao computador o que deve fazer com as informações de cada célula. Essa segunda planilha pode ser checada e corrigida sempre que for necessário.

Para entender melhor como o sistema funciona, voltemos um pouco ao nosso contador tradicional. Suponhamos que ele queira mostrar na primeira coluna o custo de um item; na segunda, a porcentagem de imposto a ser paga sobre aquele valor e, na terceira, a soma dos valores exibidos nas duas colunas anteriores. O computador pode ser programado a fim de executar essa tarefa em instantes. Colocando a instrução adequada em cada célula da coluna dois, ele multiplicará o número da coluna um por uma taxa fixa. Uma instrução semelhante em cada uma das células da coluna três fará com que ele calcule a soma das colunas um e dois.

PROJEÇÕES

Outro problema do contador que trabalha com uma planilha manual é lidar com as alterações dos dados, em geral freqüentes. Um aumento nos custos de mão-de-obra, por exemplo, pode obrigá-lo a recalcular o total de despesas e a revisar todos os valores de Lucros/Perdas. Se ele estiver simplesmente registrando valores, o problema não será tão grave. Mas, se seu trabalho incluir a projeção dados dos para um ano ou mais, ele precisará refazer centenas de cálculos.

Executada manualmente, a tarefa será demorada, cansativa e muito vulnerável a erros. Um computador poderá completá-la em alguns décimos de segundo: depois de termos colocado determinados valores na planilha, a mudança de qualquer um provocará o reajuste automático de todos os outros a ele relacionados. Se, por exemplo, o valor correspondente ao custo da matéria-prima muda, o custo total do produto é reajustado de acordo, bem como o lucro final.

Algumas planilhas são capazes de fazer cálculos bem mais complexos, prestando-se, inclusive, à solução de questões do tipo "E se...?", que aparecem constantemente no mundo dos negócios — e em muitas outras áreas. Embora empregadas principalmente para fins comerciais, elas têm utilidade em qualquer campo, sempre que se lida com muitas variáveis interdependentes.

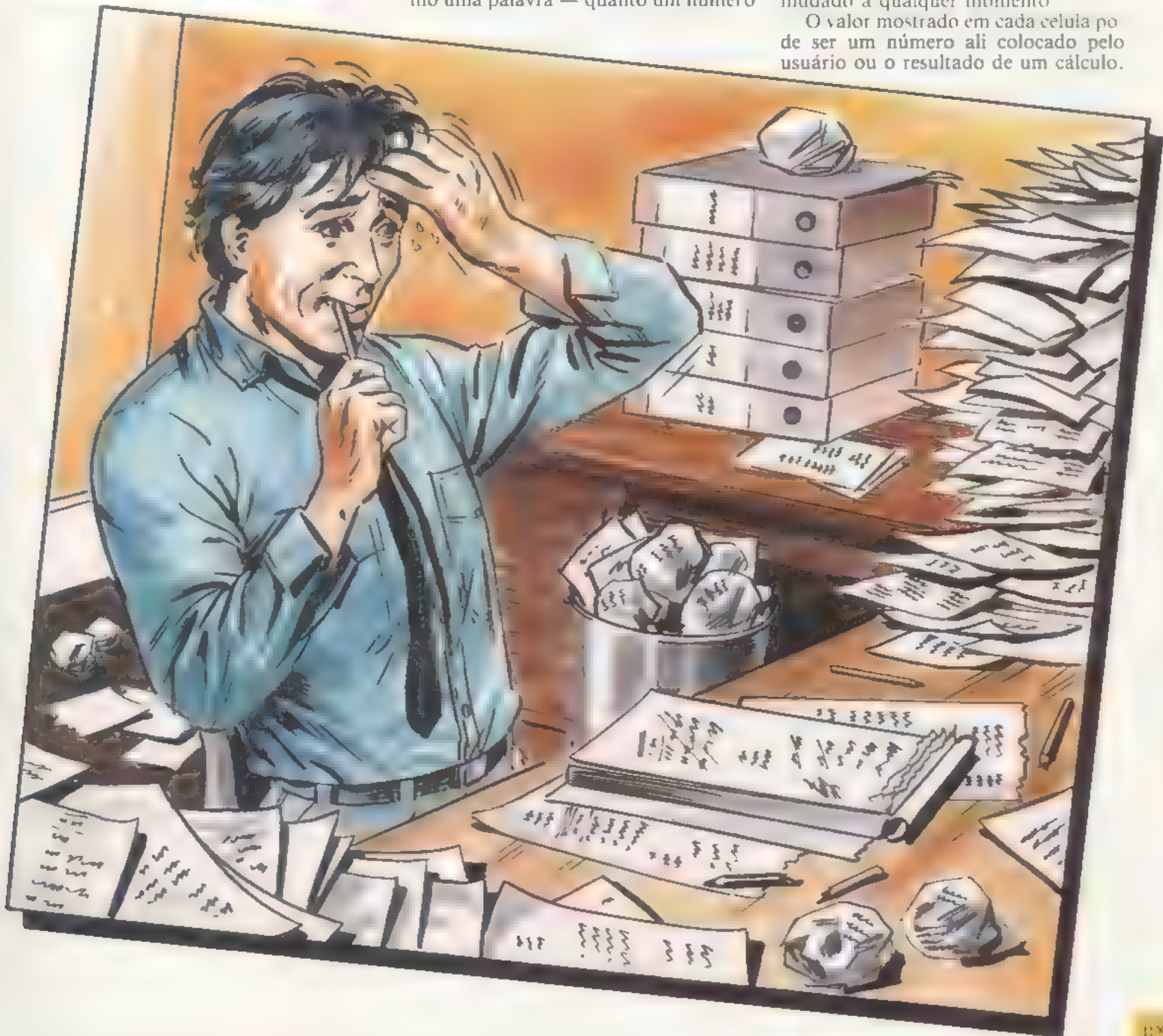
Graças à sua extraordinária versatilidade, as planilhas eletrônicas são um dos tipos mais procurados de programa aplicativo. Muitas delas são compatíveis com outros programas, possibilitando composições adequadas a aplicações de

grande complexidade. Um editor de textos, um sistema de gerenciamento de banco de dados e uma planilha eletrônica, por exemplo, constituem um poderoso conjunto de programas.

UMA PLANILHA TÍPICA

A unidade básica da planilha é a célula. O conteúdo de cada célula pode ser tanto uma variável alfanumérica — como uma palavra — quanto um número ou uma fórmula. Quando a planilha é carregada na memória do computador, as células têm um tamanho predeterminado. Em algumas delas, esse *default* (como é chamado em inglês) pode ser mudado a qualquer momento.

O valor mostrado em cada célula pode ser um número ali colocado pelo usuário ou o resultado de um cálculo.



O número de linhas e colunas varia de uma planilha para outra; as melhores possuem 65 colunas e 256 linhas, ou seja, 16640 células individuais — quantidade excessiva para um pequeno micro manipular! A planilha de INPUT tem 24 colunas e de 20 a 30 linhas, dependendo do computador.

As células são sempre identificadas por letras e números ao longo dos eixos X e Y. A maioria das planilhas usa colunas identificadas por letras: **A**, **B**, **C**, ..., **Z** e, depois, **AA**, **AB**, ... As linhas, nesse caso, são numeradas de 1 em diante. Este é o método usado no nosso programa.

Existem vários comandos disponíveis para a entrada de equações, valores, títulos, assim como para a cópia de células ou o exame de diferentes partes da planilha. Outros comandos fazem os cálculos e permitem que os dados sejam gravados ou carregados para a memória do computador. Podem-se montar equações com as operações elementares, com porcentagens ou com os totais de linhas ou colunas. No micro Spectrum, o movimento de um cursor pela planilha faz com que a célula a ser trabalhada apareça em destaque. Os demais microcomputadores usam um sistema diferente: cada célula é especificada e tem seu conteúdo exibido no rodapé da tela antes de ser transferido para a posição definitiva na planilha.

PLANEJAMENTO E DESENHO

A primeira fase de preparação de uma planilha é a mais difícil, requer detalhado planejamento e não envolve o uso do computador. Antes de mais nada, é necessário estabelecer exatamente o que se quer, pois isso afeta o desenho da planilha. Em seguida, deve-se dedicar o máximo de atenção ao planejamento, para se obter uma planilha que mostre as informações de maneira clara e concisa. Como muitas coisas em computação, esse instrumento será tão bom quanto for seu planejamento. Se você não tiver cuidado nessa fase, provavelmente obterá uma planilha confusa, difícil de ler e até com erros nas fórmulas dos cálculos.

Um exemplo prático: suponhamos que você queira desenhar uma planilha para controlar as finanças domésticas durante o ano. É evidente que os doze meses deverão encabeçar as colunas. Mas decidir o título de cada linha será um pouco mais difícil.

Primeiro: qual o nível de detalhe pretendido? Aluguel, transporte, assistência médica, alimentação são títulos óbvios quando se fala de orçamento doméstico. Mas você quer discriminar os gastos com combustível, criando uma categoria independente? Ou prefere reunir todos os gastos relacionados a transporte — além de combustível, manutenção do carro, taxas etc. — em uma categoria única? Tudo depende de seus interesses e necessidades.

Uma planilha pode ser particularmente útil para manter o valor de seus bens atualizado, fornecendo, ainda, informações como as porcentagens de valorização de sua casa ou da depreciação do seu carro.

Calcular a valorização anual de sua casa parece muito simples, inicialmente. Um ano depois de comprá-la, seu valor será o preço de compra multiplicada pela porcentagem de valorização — **P*X%**, onde **X** corresponde à porcentagem mais 100. Por exemplo, **X** seria 100 + 5% para uma valorização anual de 5%. A fórmula para calcular o valor ao fim do segundo ano seria **P*X%*X%**. Ano a ano, a fórmula fica maior e mais difícil de ser calculada.

Com uma planilha, o trabalho fica bem mais fácil, e não é preciso ser um matemático, conhecedor de dúzias de fórmulas e equações, para usar todo seu potencial. Num caso como o da valorização da casa, pode-se usar o endereço de uma célula para se referir ao conteúdo dela. Assim, a fórmula nunca fica mais complicada que **P*X%**, onde **P** é o conteúdo da célula anterior. A fórmula a ser escrita na planilha teria esta configuração: **B10%105%**.

Se a fórmula foi posta na célula **C10**, o resultado é mostrado nesse local. Colocando-se a fórmula **C10*105%** em **D10**, por exemplo, o computador toma o valor colocado em **C10** e o multiplica por 105%.

A apresentação da fórmula varia de uma planilha para outra e os programas deste artigo usam um método bem diferente dos habituais. Os detalhes serão explicados nas instruções sobre uso do programa, dadas mais tarde.

A utilização do endereço da célula em vez de seu conteúdo simplifica o trabalho com a planilha, permitindo que qualquer um, com um pouco de bom senso e paciência, realize tarefas bastante complicadas. No entanto, precisamos ter cuidado ao fazer referência a uma célula em outra. Não devemos, por exemplo, usar a célula **B10** em uma fórmula **C10**, se o resultado de **B10** depende do valor assumido por **C10**. O micro não pode calcular o valor de uma sem ter resolvido a outra! Se a planilha não detectar esse problema, o programa apresentará erro quando o computador tentar resolver o paradoxo.

E SE...?

E se as taxas de juros subirem 15% no mês de junho? E se comprarmos um carro maior? E se instalarmos um aque



cedor central? Utilizando sua planilha, você poderá obter resposta para todas essas perguntas.

Observe que a última delas aponta para outra área, que não a financeira, na qual as planilhas são úteis. A diferença entre sistemas de aquecimento central com tipos de combustível diversos pode ser visualizada num relance. Estimar a perda de calor que se evita com o uso de dupla isolação permite também prever quanto vamos economizar e quanto tempo levaremos para recuperar o investimento.

OUTROS USOS

Embora, geralmente, mesmo as planilhas mais simples sejam usadas para aplicações sérias e complicadas, elas se prestam muito bem para o lazer. Vários modelos diferentes dos financeiros podem ser construídos. Só para divertimento, crie uma referência circular com as células que não têm fim.

Existem enormes variações de uma planilha para outra. Como regra geral, quanto mais poderosa, mais cara é a planilha e maior o micro necessário para sua utilização. Uma planilha simples tem uma meia dúzia de comandos e mais ou menos o mesmo número de funções. Compare isso aos vinte comandos e quarenta funções das grandes planilhas. As mais sofisticadas permitem, inclusive, a introdução de declarações com funções similares a comandos em BASIC — por exemplo, **IF...THEN**, **AND**, **OR** e **NOT**. Em outras palavras, é possível programar as planilhas.

DIGITE O PROGRAMA

O programa da planilha eletrônica e bastante longo e, por isso, está dividido em três partes. Digite as linhas que se seguem e grave-as para depois adicionar as restantes.

O programa não funciona neste nível. As instruções de como usá-lo serão dadas nas próximas duas partes



```
5 BORDER 0· PAPER 0· INK 7·
CLS
10 DIM b$(11)· DIM s$(8)  DIM
d$(30,24,18)· DIM v(4)· DIM
z$(5,4)
20 GOSUB 1730· POKE 23658,8
LET t$="VAL"· LET os=0· LET
sflag=0· LET wx=1· LET wy=1
LET cx=1· LET cy=1
30 CLS   PRINT "
                  "· FOR x=4 TO
32 STEP 9· FOR y=2 TO 21 STEP
2· PRINT AT y,x;""· NEXT y·
NEXT x· FOR y=1 TO 21 STEP 2·
PRINT AT y,0,""· NEXT y
40 FOR x=0 TO 2: PRINT AT 0,9
*x+0,CHR$ (wx+x+64)· NEXT x
50 FOR x=0 TO 9: PRINT AT (x+
1)*2,1,(" " AND wy+x<10);wy+x
· NEXT x
60 PRINT AT 0,0;t$;" "  FOR y
=0 TO 9: FOR x=0 TO 2: GOSUB
1230: NEXT x· NEXT y· PRINT
#1;AT 0,0,"
              "
70 PRINT AT cy*2,((cx-1)*9)+5
, BRIGHT 1, FLASH 8: PAPER 8,
INK 8, OVER 1;" "
80 IF INKEY$="'" AND wy>1
THEN  LET wy wy 10· GOTO 40
90 IF INKEY$ "&" AND wy<20
THEN  LET wy wy+10· GOTO 40
100 IF INKEY$="(" AND wx<21
THEN  LET wx=wx+3  GOTO 40
110 IF INKEY$="%" AND wx>1
THEN  LET wx=wx-3· GOTO 40
120 PRINT AT cy*2,((cx-1)*9)+5
; FLASH 8: BRIGHT 0: INK 8;
PAPER 8; OVER 1;" "
130 LET cy=cy+(INKEY$="6" AND
cy<10)-(INKEY$="7" AND cy>1)
LET cx=cx+(INKEY$="8" AND cx<3
) (INKEY$="5" AND cx>1)
140 IF INKEY$ "i" OR INKEY$="I
" THEN  GOSUB 1250
150 IF INKEY$="v" OR INKEY$="V
" THEN  LET t$="VAL"· GOTO 60
160 IF INKEY$="e" OR INKEY$="E
" THEN  LET t$="IGUAL"  GOTO 6
0
170 IF INKEY$="?" THEN  PRINT
AT 0,0; FLASH 1;"CALC": GOSUB
810· GOTO 60
180 IF INKEY$="z" OR INKEY$="Z
```

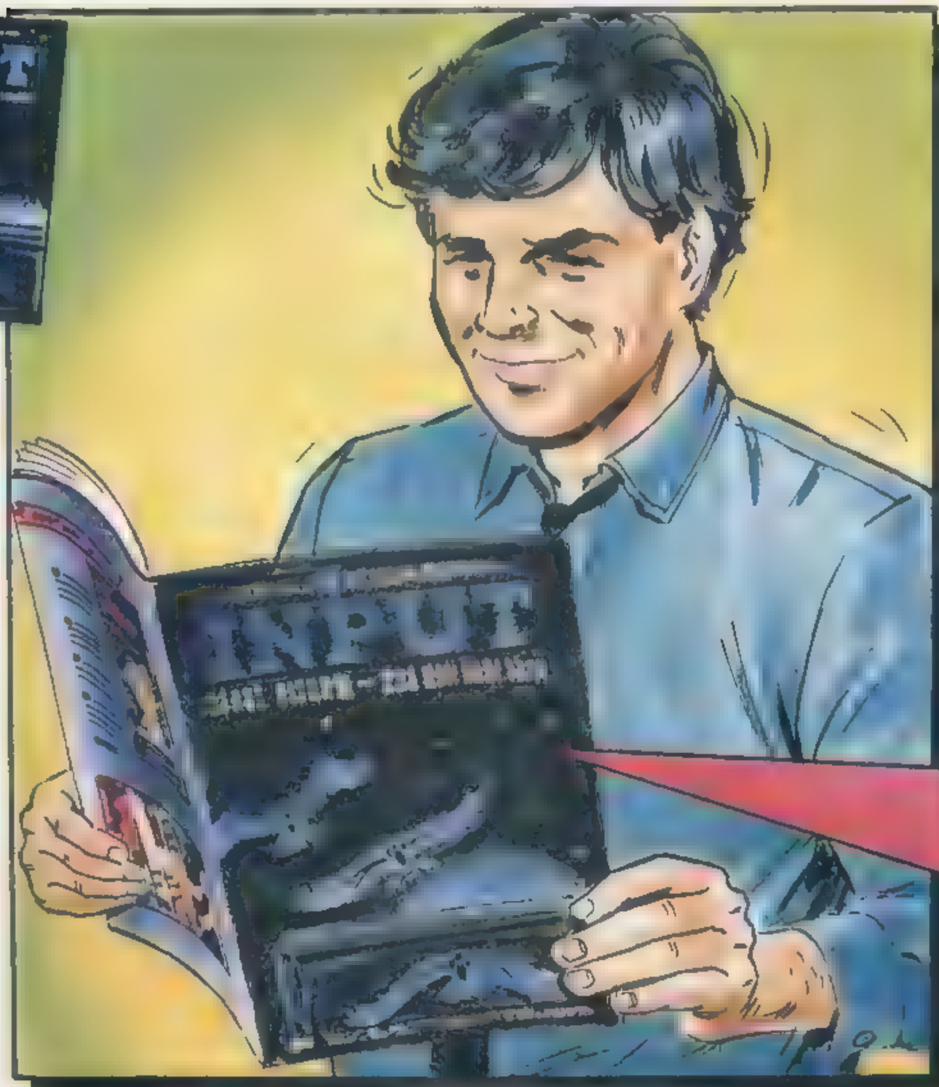
```
" THEN  PRINT AT 0,0; FLASH 1;
"COPIA": GOSUB 230· GOTO 60
190 IF INKEY$="P" OR INKEY$="p
" THEN  COPY
200 IF INKEY$="NOT " THEN
GOSUB 1780· GOTO 30
210 IF INKEY$="-" THEN  GOSUB
1840· GOTO 30
220 GOTO 70
230 PRINT #1,AT 0,0;"CELULA A
COPIAR ?"· LET d=1· LET c=3·
LET x=15· GOSUB 580· GOSUB 670
: IF f THEN  SOUND  2,30· GOTO
 230
240 PRINT #1;AT 0.0;"ABS ou RE
L (A ou R)?"· LET x=22· LET d=
2· LET c=1· GOSUB 580· GOSUB
670· IF f THEN  SOUND  2,30:
GOTO 240
250 PRINT #1;AT 0.0;"COL ou LI
NHA (C ou L)?": LET x=22: LET
d=3: LET c=1: GOSUB 580: GOSUB
 670· IF f THEN  SOUND .2,30·
GOTO 250
260 PRINT #1;AT 0.0;"DA CELULA
No  ?"· LET x=16· LET d=4: LET
c=3· GOSUB 580· GOSUB 670: IF
f THEN  SOUND  2.30  GOTO 260
270 PRINT #1;AT 0,0;"PARA A CE
LULA No. ?": LET x=14: LET d=5
: LET c=3· GOSUB 580· GOSUB
670· IF f THEN  SOUND .2,30·
GOTO 270
280 GOSUB 770· IF NOT f THEN
GOTO 320
290 PRINT #1,AT 0,0:"COMANDO E
RRADO - PRESSIONE  A   PARA AB
ORTAR OU QUALQUER OUTRA  TECLA
 PARA RE-ENTRAR"
300 PAUSE 10: PAUSE 0: PRINT #
1;AT 0,0;"

                  ": IF INKEY$="a"
OR INKEY$="A" THEN  RETURN
310 GOTO 230
320 LET a=(CODE z$(1,2))-64·
LET b=VAL z$(1,3 TO (VAL z$(1,
1)+1)): LET m$=(d$(b,a,9 TO 16
) AND t$="IGUAL")+(d$(b,a, TO
8) AND t$="VAL")· LET c$=d$(b,
a,17): LET z=CODE d$(b,a,18)
330 IF z$(2,2)="R" THEN  GOTO
 390
340 FOR a=fc TO tc· FOR b=fr
TO tr
350 IF t$="IGUAL" THEN  LET d$
(b,a,9 TO 16)=m$: LET d$(b,a,
17)=c$
360 IF t$="VAL" THEN  LET d$(b
,a, TO 8)=m$
370 NEXT b  NEXT a
380 RETURN
390 LET m$=d$(b,a,9 TO 16)·
GOSUB 890· LET a (CODE z$(4,2)
-64)-(CODE z$(1,2)-64): LET b=
VAL z$(4,3 TO (VAL z$(4,1)+1))
-VAL z$(1,3 TO (VAL z$(1,1)+1)
)
400 LET v(2)=v(2)+b-1: LET v(4
)=v(4)=((b-1) AND v,3)<>26)
410 LET v(3)=v(3)+((a-1) AND v
(3)<>26)· LET v(1)=v(1)+a 1
412 IF z$(3,2)="C" THEN  LET v
(1) v(1)+1· LET v(3)=v(1)+(v(3
)<>26)
414 IF z$(3,2)="R" THEN  LET v
(2)=v(2)+1: LET v(4)=v(4)+(v(4
)<>26)
```



```
10 PMODE 0,1 PCLEAR 1·CLEAR 100
00 CLS PRINT @230,"PLANILHA ELE
TRONICA"
20 CS=1:RS=1:CR=1:CC=1.MOS(0)="
VALOR (CALC)":MOS(1)="EQUACAO
```

```
   ":MO=1:OP$="+-*/%$&"
30 DIM D$(26,30),D(26,30)
40 FOR I=1 TO 26:FOR J=1 TO 30:
D$(I,J)=CHR$(128):NEXT J,I
50 CX=4,RX=1
60 GOSUB 70:GOTO 170
70 PRINT @448,"ESPERE":PRINT @0
,STRING$(3,128);:FOR I=CS TO CS
+3:PRINT CHR$(123);CHR$(128);CH
R$(128);CHR$(96+I);CHR$(128);CH
R$(128),CHR$(125);:NEXT:PRINT C
HR$(128);
80 PRINT @480,"MODO: ";MO$(MO);
90 FOR I=0 TO 11:C1=INT((RS+I)/
10)+48:C2=(RS+I)-((C1-48)*10)+4
8:POKE 1024+32*I+32,C1:POKE 102
4+32*I+33,C2:PRINT @32*I+34,"":
NEXT
100 PRINT @416:IF MO=0 THEN GOS
UB 740:GOTO 130
110 FOR J=RS TO RS+11:FOR I=CS
TO CS+3
120 PRINT @(J-RS)*32+35+(I-CS)*
7,"";:GOSUB 660:NEXT I,J
130 PRINT @480,"MODO: ";MO$(MO)
;TAB(20);"CELULA: ";CHR$(64+CC)
;MID$(STR$(CR),2);" ";
140 PRINT @448,"PRONTO"
150 PRINT @458,MID$(D$(CC,CR),2
)
160 RETURN
170 PS=(CR-RS+1)*32+(CC-CS)*7+3
+1024:Z=PEEK(PS):POKE PS,191 AN
D Z
180 I$=INKEY$:IF I$="" THEN 180
190 POKE PS,Z
200 IF I$=CHR$(8) AND CC>1 THEN
 CC=CC-1:IF CC<CS THEN CS=CS-1:
GOSUB 70
210 IF I$=CHR$(9) AND CC<26 THE
N CC=CC+1:IF CC>CS+3 THEN CS=CS
+1:GOSUB 70
220 IF I$=CHR$(10) AND CR<30 TH
EN CR=CR+1:IF CR>RS+11 THEN RS=
RS+1:GOSUB 70
230 IF I$=CHR$(94) AND CR>1 THE
N CR=CR-1:IF CR<RS THEN RS=RS-1
:GOSUB 70
240 IF I$="G" THEN GOSUB 330
250 IF I$="Q" THEN CLS:INPUT"CO
NFIRMA QUERER SAIR (S/N) ?";A$:
IF A$<>"S" THEN GOSUB 70 ELSE C
LS:END
260 IF I$="I" GOSUB 410
270 IF I$="V" THEN MO=0:GOSUB 7
0
280 IF I$="C" GOSUB 1490
290 IF I$="E" THEN MO=1 GOSUB 7
0
300 IF I$="S" GOSUB 1230
310 IF I$="L" GOSUB 1350
320 GOSUB 130 GOTO 170
330 PRINT @448:PRINT @448,"PARA
 A CELULA ->";:LINE INPUT A$
340 IF A$="" THEN RETURN
350 C1=ASC(A$)-64:IF C1<1 OR C1
>26 THEN 330
360 C2=VAL(MID$(A$,2)):IF C2<1
OR C2>30 THEN 330
370 CC=C1:CS=C1:CR=C2:RS=C2
380 IF CS>23 THEN CS=23
390 IF RS>19 THEN RS=19
400 GOSUB 70:RETURN
410 PRINT @448,"NOVO CONTEUDO .
";:LINE INPUT A$
420 IF A$="" THEN A$=CHR$(128)
GOTO 610
430 IF LEN(A$)>9 THEN PRINT @44
8,"ENTRADA INVALIDA":SOUND 1,4:
GOTO 410
440 IF VAL(A$)<>0 THEN 560
450 B$=LEFT$(A$,1):IF B$<"A" OR
 B$>"Z" THEN 600
460 C$=MID$(A$,2,2)
470 IF VAL(C$)<1 OR VAL(C$)>30
THEN 600
480 IF VAL(C$)<10 THEN A$=B$+ST
R$(VAL(C$))+MID$(A$,3)
490 D$=MID$(A$,4,1) IF D$<"A" O
R D$>"Z" THEN 600
500 E$=MID$(A$,5)
510 IF VAL(E$)<1 OR VAL(E$)>30
THEN 600
520 IF VAL(E$)<10 THEN A$=LEFT$
(A$,4)+STR$(VAL(E$))+MID$(A$,6)
530 O$=MID$(A$,7,1).IF INSTR(1,
OP$,O$)=0 OR O$="" THEN 600
540 DP=VAL(RIGHT$(A$,1)) IF DP<
0 OR DP>7 THEN 600
550 PRINT @448,"ENTRADA E UMA e
quacao":A$=CHR$(131)+A$.GOTO 61
0
560 PRINT @448,"ENTRADA E UM va
lor"
570 IF RIGHT$(A$,1)=" " THEN A$
=LEFT$(A$,LEN(A$)-1).GOTO 570
```

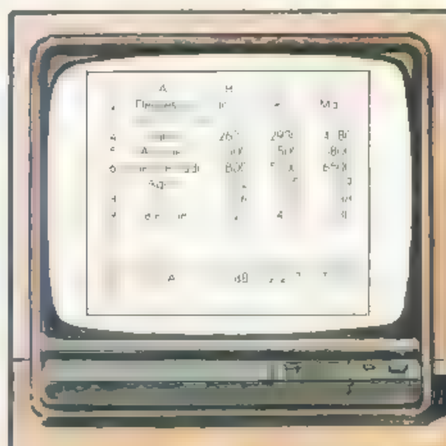




```
580 IF LEN(A$)<7 THEN A$=" "+A$
:GOTO 580
590 A$=CHR$(129)+A$:GOTO 610
600 PRINT @448,"ENTRADA E UMA 1
egenda":A$=CHR$(130)+A$
610 D$(CC,CR)=A$:I=CC:J=CR:PRIN
T @(J-RS)*32+35+(I-CS)*7,"";:GO
SUB 660:SOUND 190,2:FOR D=1 TO
500:NEXT
620 IF CC>CX THEN CX=CC
630 IF CR>RX THEN RX=CR
640 IF MO=0 THEN MO=1:GOSUB 70
650 RETURN
```



```
10 KEYOFF:CLEAR10000:COLOR15,4,
4:SCREEN0:CLS:LOCATE 9,11:PRINT
"PLANILHA ELETRONICA"
20 CS=1:RS=1:CR=1:CC=1:MO$(0)="
VALOR(CALC)":MO$(1)="EQUAÇAO
":MO=1:OP$="+-*/$$&"
30 DIM D$(26,30),D(26,30)
40 FOR I=1 TO 26:FOR J=1 TO 30:
D$(I,J)=CHR$(128):NEXTJ,I
50 CX=4:RX=1
60 GOSUB 70:GOTO 170
70 CLS:LOCATE 0,20:PRINT"AGUARD
E...":LOCATE0,0:PRINTSTRING$(3,
219);:FOR I=CS TO CS+4:PRINTCHR
$(91);SPC(2);CHR$(64+I);SPC(2);
CHR$(93);:NEXT:PRINTCHR$(219);
80 LOCATE 0,21:PRINT"MODO: ";MO
$(MO)
90 FOR I=0 TO 15:C1=INT((RS+I)/
10)+48:C2=(RS+I)-((C1-48)*10)+4
8:LOCATE 0,I+1:PRINTCHR$(C1),CH
R$(C2):NEXT
100 LOCATE 19,0:IF MO=0 THEN GO
SUB 740:GOTO 130
110 FOR J=RS TO RS+15:FOR I=CS
TO CS+4
120 LOCATE (I-CS)*7+3,J-RS+1:GO
SUB 660:NEXT I,J
130 LOCATE 0,21:PRINT"MODO: ";M
O$(MO);TAB(20)"CEL: ";CHR$(64+C
C),MID$(STR$(CR),2);"  ",
140 LOCATE 0,20:PRINT"PRONTO!
",TAB(15)MID$(D$(CC,CR),2);SPC
(9)
160 RETURN
170 PS=(CR-RS+1)*40+(CC-CS)*7+4
:Z=VPEEK(PS):VPOKE PS,219
180 I$=INKEY$:IF I$="" THEN 180
190 VPOKE PS,Z
200 IF I$=CHR$(29) AND CC>1 THE
N CC=CC-1:IF CC<CS THEN CS=CS-1
:GOSUB 70
210 IF I$=CHR$(28) AND CC<26 TH
EN CC=CC+1:IF CC>CS+4 THEN CS=C
S+1:GOSUB 70
220 IF I$=CHR$(31) AND CR<30 TH
EN CR=CR+1:IF CR>RS+15 THEN RS=
RS+1:GOSUB 70
230 IF I$=CHR$(30) AND CR>1 THE
N CR=CR-1:IF CR<RS THEN RS=RS-1
:GOSUB 70
240 IF I$="G" THEN GOSUB 330
250 IF I$="Q" THEN CLS:INPUT"TE
RMINO O PROGRAMA? (S/N) ";A$:IF
 A$<>"S" THEN GOSUB 70 ELSECLS:
 END
260 IF I$="I" THEN GOSUB 410
270 IF I$="V" THEN MO=0:GOSUB 7
0
280 IF I$="C" THEN GOSUB 1490
290 IF I$="E" THEN MO=1:GOSUB 7
0
300 IF I$="S" THEN GOSUB 1230
310 IF I$="L" THEN GOSUB 1350
320 GOSUB 130:GOTO 170
330 LOCATE 0,20:PRINTSPC(38):LO
CATE 0,20:PRINT"PARA CEL >",:IN
PUT A$
340 IF A$="" THEN RETURN
350 C1=ASC(A$)-64:IF C1<1 OR C1
>26 THEN 330
360 C2=VAL(MID$(A$,2)):IF C2<1
OR C2>26 THEN 330
370 CC=C1:CS=C1:CR=C2:RS=C2
```



```
380 IF CS>23 THEN CS=23
390 IF RS>19 THEN RS=19
400 GOSUB 70:RETURN
410 LOCATE 0,20:PRINT"NOVO CONT
EUDO: ";SPC(22);:LOCATE 15:LINE
INPUT A$
420 IF A$="" THEN A$=CHR$(128):
GOTO 610
430 IF LEN(A$)>9 THEN LOCATE 0,
20:PRINT"ENTRADA INVALIDA
":FOR I=1 TO 500:NEXT:GOTO 410
440 IF VAL(A$)<>0 THEN 560
450 B$=LEFT$(A$,1):IF B$<"A" OR
B$>"Z" THEN 600
460 C$=MID$(A$,2,2)
470 IF VAL(C$)<1 OR VAL(C$)>30
THEN 600
480 IF VAL(C$)<10 THEN A$=B$+ST
R$(VAL(C$))+MID$(A$,3)
490 D$=MID$(A$,4,1):IF D$<"A" O
R D$>"Z" THEN 600
500 E$=MID$(A$,5)
510 IF VAL(E$)<1 OR VAL(E$)>30
THEN 600
520 IF VAL(E$)<10 THEN A$=LEFT$
(A$,4)+STR$(VAL(E$))+MID$(A$,6)
530 O$=MID$(A$,7,1):IF INSTR(1,
OP$,O$)=0 OR O$="" THEN 600
540 DP=VAL(RIGHT$(A$,1)):IF DP<
0 OR DP>7 THEN 600
550 LOCATE 0,20:PRINT"A entrada
 é uma EQUAÇAO":A$=CHR$(131)+A$
:GOTO 610
560 LOCATE 0,20:PRINT"A entrada
 é um VALOR"
```



```
570 IF RIGHT$(A$,1)=" " THEN A$
=LEFT$(A$,LEN(A$)-1):GOTO 570
580 IF LEN(A$)<7 THEN A$=" "+A$
:GOTO 580
590 A$=CHR$(129)+A$:GOTO 610
600 LOCATE 0,20:PRINT"A entrada
 é um RÓTULO":A$=CHR$(130)+A$
610 D$(CC,CR)=A$:I=CC:J=CR:LOCA
TE (I-CS)*7+3,(J-RS)+1:GOSUB 66
0
620 IF CC>CX THEN CX=CC
630 IF CR>RX THEN RX=CR
640 IF MO=0 THEN MO=1:GOSUB 70
650 RETURN
```



```
10  TEXT : HOME : CLEAR : VTAB
12: HTAB 10: PRINT "PLANILHA EL
ETRONICA"
20 CS = 1:RS = 1:CR = 1:CC = 1:
CA = 1:RA = 1:MO$(0) = "VALOR(C
ALC)":MO$(1) = "EQUACAO     ":M
O = 1:OP$ = "+-*/t$&":D$ =  CHR
$ (13) +  CHR$ (4)
25  ONERR  GOTO 2500
30  DIM D$(26,30),D(26,30)
40  FOR I = 1 TO 26: FOR J = 1
TO 30:D$(I,J) =  CHR$ (128): NE
XT : NEXT
50 CX = 4:RX = 1: FOR I = 1 TO
10:LL$ = LL$ +  CHR$ (255): NEX
T
60  GOSUB 70: GOTO 170
70  HOME : VTAB 2: PRINT "AGUA
RDE     ": VTAB 1: PRINT  LEFT$ (
LL$,3);: FOR I = CS TO CS + 4:
PRINT  CHR$ (91); SPC( 2); CHR$
 (64 + I); SPC( 2); CHR$ (93);
 NEXT : PRINT  LEFT$ (LL$,2)
80  VTAB 22: PRINT "MODO: ";MO$
(MO);
90  INVERSE : HTAB 1: FOR I = 0
 TO 15:C1 =  INT ((RS + I) / 10
) + 48:C2 = (RS + I) - ((C1 - 4
8) * 10) + 48: VTAB I + 2: PRIN
T  CHR$ (C1); CHR$ (C2): NEXT :
 NORMAL
100  VTAB 19: IF MO = 0 THEN  G
OSUB 740: GOTO 130
110  FOR J = RS TO RS + 15: FOR
 I = CS TO CS + 4
120  VTAB J - RS + 2: HTAB (I -
 CS) * 7 + 4: GOSUB 660: NEXT :
 NEXT
130  VTAB 22: HTAB 1: PRINT "MO
DO: ";MO$(MO); TAB( 20);"CEL: "
; CHR$ (64 + CC);CR;" "
140  VTAB 21: HTAB 1: CALL  - 8
68: PRINT "PRONTO!"; TAB( 15);D
$(CC,CR)
160  RETURN
170 V = CR - RS + 2:H = (CC - C
S) * 7 + 4:VA = RA - RS + 2:HA
= (CA - CS) * 7 + 4
180  VTAB VA: HTAB HA:I = CA:J
= RA:.GOSUB 660
182  VTAB V: HTAB H: INVERSE :I
 = CC:J = CR: GOSUB 660
183  PRINT : NORMAL
185 CA = CC:RA = CR
190  GET I$
200  IF I$ =  CHR$ (8) AND CC >
 1 THEN CC = CC - 1: IF CC < CS
 THEN CS = CS - 1: GOSUB 70
210  IF I$ =  CHR$ (21) AND CC
< 26 THEN CC = CC + 1: IF CC >
CS + 4 THEN CS = CS + 1: GOSUB
70
220  IF I$ =  CHR$ (90) AND CR
< 30 THEN CR = CR + 1: IF CR >
RS + 15 THEN RS = RS + 1: GOSUB
 70
230  IF I$ =  CHR$ (65) AND CR
> 1 THEN CR = CR - 1: IF CR < R
S THEN RS = RS - 1: GOSUB 70
240  IF I$ = "G" THEN  GOSUB 33
0
250  IF I$ = "Q" THEN  GOSUB 30
00
260  IF I$ = "I" THEN  GOSUB 41
0
270  IF I$ = "V" THEN MO = 0: G
OSUB 70
280  IF I$ = "C" THEN  GOSUB 14
90
290  IF I$ = "E" THEN MO = 1: G
OSUB 70
300  IF I$ = "S" THEN  GOSUB 12
30
310  IF I$ = "L" THEN  GOTO 135
0
320  GOSUB 130: GOTO 170
330  VTAB 21: HTAB 1: CALL  - 9
58: PRINT "PARA CEL >";: INPUT
A$
340  IF A$ = "" THEN  RETURN
350 C1 =  ASC (A$) - 64: IF C1
< 1 OR C1 > 26 THEN 330
360 C2 =  VAL ( MID$ (A$,2)): I
F C2 < 1 OR C2 > 30 THEN 330
370 CC = C1:CS = C1:CA = C1:CR
= C2:RS = C2:RA = CR
380  IF CS > 23 THEN CS = 23
390  IF RS > 19 THEN RS = 19
400  GOSUB 70: RETURN
410  VTAB 21: HTAB 1: CALL  - 9
58: PRINT "NOVO CONTEUDO: ";: I
NPUT A$
420  IF A$ = "" THEN A$ =  CHR$
 (128): GOTO 610
430  IF  LEN (A$) > 9 THEN  VTA
B 20: PRINT "PALAVRA MUITO GRAN
DE!"; CHR$ (7): FOR X = 1 TO 30
0: NEXT : GOTO 410
440  IF  VAL (A$) <  > 0 THEN 5
60
450 B$ =  LEFT$ (A$,1): IF B$ <
 "A" OR B$ > "Z" THEN 600
460 C$ =  MID$ (A$,2,2)
470  IF  VAL (C$) < 1 OR  VAL (
C$) > 30 THEN 600
480  IF  VAL (C$) < 10 THEN A$
= B$ + " " +  MID$ (A$,2)
490 DD$ =  MID$ (A$,4,1): IF DD
$ < "A" OR DD$ > "Z" THEN 600
500 E$ =  MID$ (A$,5)
510  IF  VAL (E$) < 1 OR  VAL (
E$) > 30 THEN 600
520  IF  VAL (E$) < 10 THEN A$
=  LEFT$ (A$,4) + " " +  MID$ (
A$,5)
530 O$ =  MID$ (A$,7,1): FOR D
= 1 TO  LEN (OP$): IF O$ <  >
MID$ (OP$,D,1) THEN  NEXT : GOT
O 600
540 DP =  VAL ( RIGHT$ (A$,1)):
 IF DP < 0 OR DP > 7 THEN 600
550  VTAB 21: HTAB 1: CALL  - 9
58: PRINT "A entrada e uma EQUA
CAO":A$ =  CHR$ (131) + A$: GOT
O 610
560  VTAB 21: HTAB 1: CALL  - 9
58: PRINT "A entrada e um VALOR
"
570  IF  RIGHT$ (A$,1) = " " TH
EN A$ =  LEFT$ (A$, LEN (A$) -
1): GOTO 570
580  IF  LEN (A$) < 7 THEN A$ =
 " " + A$: GOTO 580
590 A$ =  CHR$ (129) + A$: GOTO
 610
600  VTAB 21: HTAB 1: CALL  - 9
58: PRINT "A entrada e um ROTUL
O":A$ =  CHR$ (130) + A$
610 D$(CC,CR) = A$: FOR D = 1 T
O 500: NEXT
620  IF CC > CX THEN CX = CC
630  IF CR > RX THEN RX = CR
640  IF MO = 0 THEN MO = 1: GOS
UB 70
650  RETURN
```



Como as planilhas tradicionais dos contadores, nossa planilha de cálculo eletrônica tambem está dividida em celulas formadas por colunas, que correspondem aos meses do ano, e linhas, equivalentes a rótulos diversos, como, por exemplo, receitas e despesas. Sua grande vantagem reside na forma de manipulação dos dados, que podem ser calculados, checados e alterados com rapidez.

AVALANCHE: AS PEDRAS ROLAM (1)

Mais e mais problemas se abatem sobre Willie. Além das serpentes assassinas, dos buracos e da alta da maré, pedras gigantescas rolam morro abaixo, ameaçando soterrá-lo.

Nosso personagem está em grandes dificuldades: enquanto os cabritos monteses devoram seu lanche, ele tenta escalar o penhasco, para não morrer afogado nas águas do mar. Por todo o caminho, buracos e cobras ameaçam sua vida. E, para completar, faremos com que ele se veja diante de uma avalanche de gigantescas pedras!

Teremos bastante trabalho na criação deste novo problema. Precisaremos colocar uma pedra no topo da encosta, empurrá-la morro abaixo, animá-la para que pareça estar rolando, verificar se ela atingiu Willie e, finalmente, apagá-la, quando chegar ao mar. Como esse processo é um tanto complexo, vamos apresentá-lo em duas partes.

A primeira parte da rotina movimenta a pedra e dá início à sua descida encosta abaixo. A segunda verifica se Willie foi atingido e coloca a pedra no topo da encosta. Apresentamos aqui apenas a primeira parte. Digite-a, mas não a execute ainda. Incompleta, a rotina não funcionará.

```
 10 REM org 58993
 20 REM bar ld a,(57344)
 30 REM cp 0
 40 REM jr z,blm
 50 REM cp 3
 60 REM jr z,blm
 70 REM ret
 80 REM blm ld a,(57358)
 90 REM cp 1
100 REM jr z,bma
110 REM ld hl,(57356)
120 REM ld bc,57120
130 REM ld a,42
140 REM call 58217
150 REM inc hl
160 REM ld a,45
170 REM ld bc,15616
180 REM call 58217
190 REM ld hl,(57356)
200 REM ld de,480
210 REM sbc hl,de
220 REM jr z,bri
230 REM ld hl,(57356)
240 REM ld de,22560
250 REM add hl,de
260 REM ld a,(hl)
270 REM cp 15
280 REM jr z,bri
290 REM cp 45
300 REM jr nz,bok
310 REM ld hl,(57356)
320 REM ld bc,15616
330 REM ld a,45
340 REM call 58217
350 REM ld de,32
360 REM add hl,de
370 REM ld (57356),hl
380 REM ld bc,57120
390 REM ld a,42
400 REM call 58217
410 REM bok ld hl,(57356)
420 REM dec hl
430 REM ld (57356),hl
440 REM ld a,1
450 REM ld (57358),a
460 REM ret
470 REM org 59137
480 REM bri *
490 REM org 59097
500 REM bma *
```

Antes de mais nada, a rotina de movimentação da pedra verifica se a avalanche pode ocorrer no nível atual do jogo. Como você deve se lembrar, Willie tenta não ser alcançado por pedras, saltando sobre elas, no nível um e no nível quatro. A variável correspondente ao ní-



vel atual do jogo é armazenada na posição 57344 da memória; o valor 0 indica o nível um; o valor 3 aponta o nível quatro.

Assim, o conteúdo do endereço 57344 é carregado no acumulador e comparado inicialmente com o valor 0, e, depois, com 3. Se qualquer um desses valores estiver presente, a instrução **jr z,blm** faz com que o processador salte para a rotina de movimentação da pedra. Caso contrário, o processador encontra a instrução **ret** e retorna para a rotina principal do jogo.



QUAL DAS PEDRAS?

Para que possamos animar a pedra, existem na memória dois conjuntos de dados para dois diferentes desenhos da pedra. Quando são impressos alternadamente na tela, eles dão a impressão de que a pedra está rolando.

O processador precisa saber qual foi o último desenho impresso na tela, para poder selecionar adequadamente o próximo. Obtém essa informação no endereço 57358, onde há uma variável cujo conteúdo é 0 ou 1. Esse valor é carregado no acumulador e, se for igual a 1, o processador salta para a rotina **bma**, que será dada mais tarde. Caso o conteúdo do endereço 57358 seja igual a 0, o processador prossegue com a rotina aqui apresentada.

Como você poderá perceber, a variável no endereço 57358 oscilará — se seu conteúdo for 1, será trocado para 0 e vice-versa. Assim, quando o processador voltar a utilizar a rotina de movimentação da pedra, ele irá executar a parte do programa que deixou de lado na vez anterior, imprimindo o outro desenho da pedra.

Naturalmente, o primeiro valor do conteúdo de 57358 é definido pela rotina de inicialização incumbida de ajustar o andamento do jogo.

IMPRIMIR E APAGAR

Os endereços 57356 e 57357 guardam a posição da pedra, cujo valor é carregado no par HL. O par BC é carregado com 57120, que corresponde ao início dos dados do primeiro desenho da pedra. O código 42 (vermelho sobre fundo azul ciano) é carregado em A.

A rotina **print**, que começa no endereço 57217, é chamada. Como de costume, ela imprime os dados apontados pelo conteúdo de BC com a cor especificada pelo conteúdo de A, na posição dada pelo conteúdo de HL.

O par HL é incrementado e passa a apontar para a posição à direita da pedra. Por enquanto, estamos fazendo apenas uma pedra rolar numa parte plana da montanha. Mais tarde, você verá como utilizar um deslocamento de um espaço para simular a descida da pedra pela encosta.

O acumulador A é carregado com 45 (código da cor ciano sobre fundo ciano) e o par BC, com 15616. Esta posição está na ROM e inicia os dados para um espaço vazio. A rotina **print** é, então, chamada, de novo. A operação faz com que a pedra anteriormente impressa seja apagada. Sem isso, veríamos na tela uma fileira contínua de pedras.

FIM DA LINHA

A próxima tarefa consiste em verificar se a pedra atingiu o canto esquerdo da tela — nessa direção, ela pode ir além da posição 480. Em seguida, o par HL, já incrementado, é recarregado com a posição armazenada em 57356 e 57357. O par DE é carregado com 480 e esse valor é subtraído de HL.

Se o resultado for igual a zero, a pedra está na posição 480 — ou seja, chegou ao fim da descida. A instrução **jr z,bri** manda, então, o processador para a rotina **bri**. Esta, por sua vez, apaga a pedra do final da encosta e reajusta sua posição, mandando-a de volta para o topo da montanha.

Uma vez que não apresentaremos a rotina **bri** no momento, você precisará esperar um pouco mais para ver esta parte do programa funcionar.

CHEGOU AO MAR?

Como a maré está subindo, a pedra poderá chegar à água antes de alcançar o canto esquerdo da tela. Assim, será preciso verificar se a pedra afundou. Caso tenha se chocado contra a água, deverá ser colocada no topo de novo.

O método mais simples para verificar se a pedra atingiu a água consiste em analisar a cor da posição de tela imediatamente abaixo dela. Se for branco sobre azul, a cor do mar, a pedra deve ser apagada. O par de registros HL é novamente carregado com a posição da pedra que se encontra nos endereços 57356 e 57357. O par DE, por sua vez, é carregado com 22560. Na verdade, 22528

posições de memória separam a posição na tela (que está no arquivo de vídeo) da cor que lhe corresponde (que está no arquivo de cores). O valor 32 é adicionado a esse número para que o endereço da cor da posição de tela na linha abaixo — ou 32 caracteres depois — seja localizado.

Os conteúdos de HL e DE são somados em seguida. Sempre se utilizam esses pares de registro para fazer adições ou subtrações entre números de dois bytes, guardando-se o resultado em HL. Portanto, o conteúdo da posição de memória apontada pelo par HL — que agora equivale à posição adequada no arquivo de cores — é carregado no acumulador por meio da instrução de endereçamento indireto **ld a,(hl)**

Esse valor é, então, comparado a 15, que é o código de branco sobre azul, ou seja, a cor do mar. Se o mar estiver abaixo da pedra, a instrução **jr,zbri** manda o processador para a rotina que apaga a pedra e a coloca de volta no topo da encosta.

RECONHECIMENTO DO DECLIVE

Como já temos no acumulador a cor do caractere abaixo da pedra, vamos aproveitar para verificar se ela está ou não no chão. Para isso, o conteúdo do acumulador é comparado com 45, código correspondente à cor do céu. Se os dois valores forem iguais, esta cor está por baixo da pedra, e o processador continua com a parte seguinte da rotina. Caso contrário, a pedra ainda está firme no chão, e a instrução **jr nz,bok** faz o processador pular para o rótulo **bok**. Este deixa a pedra onde ela está, e simplesmente acerta as variáveis antes de retornar.

Você deve ter reparado que, até agora, a pedra só tem se movido para a esquerda. Sua posição vertical não foi alterada — ou seja, o declive da encosta não foi levado em conta. Portanto, a pedra só pode estar fora do chão (com a cor do céu sob ela) se tiver passado por uma seção inclinada com um caractere à esquerda — o que significa que precisamos apagá-la e imprimi-la um caractere abaixo. Em linguagem de máquina, tudo isso é feito tão rapidamente que nosso olho — ou a tela de TV — não tem tempo de reagir. Assim, você não poderá ver a pedra em sua posição intermediária, indo para o ar ou voltando para o chão.

Depois que o par de registros HL é carregado com a posição atual da pedra no vídeo, através das posições 57356 e 57357, o par BC é carregado com os dados da ROM para um espaço vazio, como antes. O acumulador A, por sua vez, é carregado com 45 — ciano sobre ciano. A rotina **print** é, então, chamada para apagar a pedra.

O par DE é carregado com 32, valor que se adiciona ao conteúdo HL, o que faz o apontador nesse par de registros mover-se uma posição para baixo na tela. Para ajustar a posição da pedra, o apontador em HL é copiado de volta nos endereços 57356 e 57357. Lembre-se de que a instrução **ld** apenas copia o conteúdo de um endereço ou de um registro em outro endereço ou registro. O conteúdo do lugar de partida permanece inalterado. Logo, quando BC é carregado com o endereço inicial dos padrões da primeira figura da pedra, a nova posição na tela ainda está em HL. O acumulador é carregado com 42 — vermelho sobre ciano — e a rotina **print** é chamada, imprimindo uma pedra vermelha uma posição abaixo da posição anterior de impressão.

ROLANDO PELA ENCOSTA

Quando a pedra é novamente impressa — não importa se uma posição abaixo ou não —, as variáveis têm que ser reajustadas para fazer com que ela pareça estar rolando morro abaixo.

Em seguida, o par HL volta a ser carregado com a posição atual da pedra, pois a rotina **bok** pode ter sido chamada diretamente, não se executando a parte do programa que muda a figura de lugar. Depois de decrementado, o conteúdo de HL é carregado em 57356 e 57357. Na próxima chamada da rotina de movimentação, esse apontador indicará uma posição à esquerda.

Se o conteúdo da variável do tipo de pedra — em 57358 — é 0, o processador continua nesse laço da rotina e imprime a primeira figura. Para que a segunda figura seja impressa, 1 é carregado em 57358 pelo acumulador.



A primeira parte da rotina de movimentação da pedra, aqui apresentada, inicia o deslocamento da figura e verifica se ela colide com alguma coisa. Essa rotina não funcionará sem a segunda parte, que continua imprimindo a pedra na tela. Assim, por enquanto, apenas digite e monte estas linhas; não tente executá-las.

```
10  ORG 19789
20  BAR LDA 18238
30  BEQ BLM
40  CMPA #3
50  BEQ BLM
60  RTS
70  BLM LDX 18253
80  LDU #1536
90  PSHS X
100  JSR CHARPR
110  PULS X
120  LEAX -1,X
130  CMPX #5344
140  BEQ BRI
150  STX 18253
160  LDA,X
170  CMPA #$AA
180  BEQ BRI
190  CMPA #$55
200  BEQ BNH
210  CMPA #$50
220  BEQ BNH
230  LDA #2
240  STA 18252
250  BNH LEAX 289,X
260  LDA ,X
270  CMPA #$AA
280  BEQ BRI
290  CMPA #$55
300  BNE BOK
310  LEAX -33,X
320  STX 18253
330  CHARPR EQU 19402
340  BOK EQU 19861
350  BRI EQU 19894
```

Antes de mais nada, a rotina de movimentação da pedra verifica se a avalanche pode ocorrer no nível atual do jogo. Como você deve se lembrar, Willie procura não ser atingido pelas pedras, saltando sobre elas, no nível um e no nível quatro. A variável correspondente ao nível do jogo está armazenada na posição de memória 18238; o valor 0 indica o nível um; o valor 3 aponta o nível quatro. Assim, o conteúdo do endereço 18238 é carregado no acumulador; se o valor for 0 ou 3 a instrução **BEQ** faz o processador pular diretamente para a rotina de movimentação da pedra. Caso contrário, o processador encontra a instrução **RET** e retorna para a rotina principal do jogo.

APAGUE A PEDRA

Na posição de memória 18253 está a variável que contém a posição atual da pedra. Essa posição é carregada no registro X, e o número 1536, no registro U. Como U é o apontador da pilha do usuário, a região da memória situada acima do endereço 1536 passa a ser, para todos os fins, a pilha do usuário. A posição 1536 pertence à memória de tela, correspondendo a uma parte do céu. O processador pula, então, para a sub-rotina **CHARPR**

Lembre-se de que essa sub-rotina uti-

liza os dados da pilha do usuário e imprime na tela um byte de cada vez (um caractere apontado por X). Logo, uma parte do céu é impressa sobre a pedra, apagando-a da tela.

Observe que, antes da rotina **CHARPR** ter sido chamada, o conteúdo de X foi guardado na pilha, mas não removido do registrador — seu valor foi simplesmente copiado. Procedemos assim porque a rotina **CHARPR** pode interferir com o registro X, já que imprime oito bytes de dados que formam um caractere. Ao chamar uma sub-rotina, convém sempre guardar na pilha o conteúdo de um registro que precisamos preservar. A regra é: na dúvida, empilhe, você evitará vários erros.

FIM DA ENCOSTA

Para obter de volta a posição na tela, basta recuperá-la da pilha da máquina. A instrução **LEAX -1,X** decrementa o conteúdo de X, que passa a apontar para uma posição à esquerda. Esse valor é comparado com 5344, endereço da posição do canto esquerdo da tela, onde a pedra irá bater após ter descido toda a encosta.

Se o valor de X for 5344, a pedra atingiu o canto da tela. A instrução **BEQ BRI** manda, então, o processador para a rotina **BRI**, que leva a pedra de volta para o topo da montanha.

Se X não for igual a 5344, a pedra não chegou ao canto da tela. Nesse caso, a instrução **STX 1823** armazena a próxima posição de impressão — que é um caractere à esquerda da anterior na variável de posição da pedra, que está em 1823. É nesta nova posição que a pedra será impressa.

WILLIE FOI ATINGIDO?

Precisamos, também, averiguar se Willie foi atingido pela pedra. Assim, antes que a nova pedra seja impressa, o conteúdo da posição de tela apontada por X é carregado no acumulador por meio da instrução **LDA ,X**. Em seguida, esse valor é comparado com $55, que é o código de amarelo, a cor do céu, e com $50, a cor da língua da cobra. Se o caractere contém $55 ou $50 — sendo, portanto, parte do céu ou da cobra — o processador pula as duas próximas instruções. Caso contrario, a pedra deve ter atingido Willie — ele é a única figura que pode estar em seu caminho. O salto então não ocorre. O acumulador é carregado com 2. Esse valor é armazenado na variável chamada **dead**, que indica se Willie morreu ou não. Ela será verificada mais tarde, em outra rotina do jogo.

A PEDRA AFUNDA

A tarefa seguinte consiste em checar se a pedra alcançou a superfície da água. Em caso afirmativo, não será preciso imprimir uma pedra ali: ela terá que ser colocada, novamente, no topo da encosta.

O conteúdo do registro X é adicionado a 289, para indicar a próxima posição na tela, uma linha abaixo. O número 289 resulta da operação 32 x 8 + 32 + 1 — ou seja, para chegar à posição imediatamente inferior, devemos contar oito linhas de 32 bytes; como a pedra ocupa uma linha de pixels acima do chão, somamos 32 bytes ao longo da memória de tela; finalmente, adicionamos o número 1 para compensar a subtração anteriormente feita para deslocar o apontador de tela uma posição para a esquerda.

O valor contido nesse caractere é carregado no acumulador por intermédio da instrução **LDA ,X** e, em seguida, comparado com $AA, que é o código correspondente a azul, a cor do mar. Se o mar está nesse caractere, a instrução **BEQ BRI** salta o processador sobre a rotina que inicializa a volta da pedra à sua posição no topo da encosta.

POSIÇÃO

Já que é preciso examinar a posição de tela imediatamente inferior, podemos muito bem verificar se existe céu naquela posição. Para isso, basta comparar o seu valor com $55, o código da cor do céu.

Caso não haja céu onde está a pedra — ou seja, se ela está no chão —, a instrução **BNE BOK** salta para a rotina **BOK**, que imprime a pedra.

Porém, se existe céu por baixo da pedra, ela será movida uma posição para baixo e uma posição para a esquerda. Lembre-se de que agora o apontador de tela está indicando uma linha de pixels abaixo da última posição da pedra. Logo, devemos subtrair o valor 32 para mover uma linha de pixels, e o valor 1, para mover uma posição para a esquerda. Isso é feito de uma só vez pela instrução **LEAX -33,X**.

Para imprimir a pedra no lugar adequado, o processador trabalha diretamente com a rotina **BOK**. Essa rotina, bem como a rotina **BRI**, será exposta num próximo artigo.

A parte da rotina de movimentação da pedra que apresentamos a seguir verifica o nível do jogo e imprime a primeira figura da pedra e, se for o caso, faz com que ela role morro abaixo. Além disso, checa se a pedra chegou ao fim da encosta ou se já alcançou a superfície do mar. A segunda parte da rotina, que daremos num artigo futuro, verifica se Willie foi atingido, imprime a segunda pedra e recoloca a figura no topo da encosta.

Digite e monte a primeira parte, mas não a execute,pois a rotina só funcionará quando estiver completa.

```
10    org 54400
20    ld a,(-5228)
30    cp 0
40    jr z,bl
50    cp 3
60    jr z,bl
70    ret
80    bl ld a,(-5195)
90    cp 1
100    jr z,bm
110    ld hl,(62407)
120    ld de,(-5200)
130    add hl,de
140    ld a,13
150    push hl
160    call 77
170    pop hl
180    inc hl
190    ld a,255
200    call 77
210    ld hl,(-5200)
220    ld de,480
230    sbc hl,de
240    jr z,mo
250    ld hl,(62407)
260    ld de,(-5200)
270    add hl,de
280    ld de,32
290    add hl,de
300    call 74
310    cp 72
320    jr z,mo
330    cp 76
340    jr z,mo
350    cp 255
360    jr nz,bo
370    ld hl,(62407)
380    ld de,(-5200)
390    add hl,de
400    ld a,255
410    push de
420    call 77
430    pop de
440    ld hl,32
450    add hl,de
460    ld (-5200),hl
470    ld de,(62407)
480    add hl,de
490    ld a,13
500    call 77
510    bo ld hl,(-5200)
520    dec hl
```

```
530    ld (-5200),hl
540    ld a,l
550    ld (-5195),a
560    ret
570    bm call -11006
580    ret
590    mo call -10953
600    ret
610    end
```

Antes de mais nada, a rotina verifica se a avalanche de pedras é necessária no nível atual do jogo. Lembre-se de que Willie tenta escapar das pedras nos níveis um e quatro. A variável que indica o nível do jogo está em -5228 — o valor 0 corresponde ao nível um e o valor 3, ao nível quatro.

O conteúdo desse endereço é carregado no acumulador e comparado inicialmente com 0 e, depois, com 3. Se algum desses valores estiver presente, a instrução **jr z,bl** faz o processador saltar para a rotina de movimentação da pedra. Caso contrário, o processador encontra a instrução **ret** e retorna ao programa principal do jogo.

QUAL DAS PEDRAS?

Para criar o efeito de movimento, utilizamos dois padrões diferentes para a pedra, imprimindo-os alternadamente. O processador precisa saber qual foi a última figura impressa para poder selecionar adequadamente a seguinte. Obtém essa informação no endereço -5195, onde existe uma variável cujo conteúdo é 0 ou 1. Esse valor é carregado no acumulador e, se for igual a 1, o processador salta para a rotina **bm** que será apresentada mais tarde. Caso contrário, ele prossegue com a rotina exposta aqui.

Como você vai perceber, o valor de -5195 é trocado sempre que se chama a rotina de movimentação. Portanto, o processador executa uma parte do programa de cada vez.

IMPRESSÃO

A posição da pedra é armazenada nos endereços -5200 e -5199. Seu valor é carregado no par DE. O endereço inicial da Tabela de Nomes (TN), que está armazenado nos endereços 62407 e 62408, é carregado em HL. A soma dos valores nesse par de registros fornece o valor adequado na TN.

O acumulador A é carregado com o código do padrão que forma o primeiro desenho da pedra. Como você verá na segunda parte da rotina, a outra pedra é formada por dois padrões. Em seguida, a rotina 77 da ROM é chamada, encarregando-se de colocar o valor contido em A no endereço da VRAM apontado pelo par HL — ou seja, a rotina imprime a pedra na tela do computador.

FIM DA ENCOSTA

A etapa seguinte consiste em verificar se a pedra atingiu o fim da encosta. Para isso, sua posição é transferida de -5200 e -5199 para o par HL e o valor 480 é carregado em DE, que representa a última posição que a pedra pode ocupar. Esse valor é subtraído de HL. Se o resultado for zero, a pedra está na posição 480 — em outras palavras, chegou ao fim da encosta. A instrução **jr z,mo** manda, então, o processador para a rotina **mo**, que apresentaremos mais tarde. Essa rotina encarrega-se de apagar a pedra e levá-la de volta para o topo da montanha.

A PEDRA AFUNDOU?

Como a maré está subindo rapidamente, é provável que a pedra atinja a água antes de chegar ao fim do morro.

Precisamos verificar se ela afundou. Em caso afirmativo, a pedra será recolocada no topo da encosta.

A rotina 74 da ROM cuida disso. Se a chamarmos com um endereço da VRAM em HL, ela devolverá o conteúdo desse endereço ao acumulador.

Portanto, carregamos o endereço inicial da TN da VRAM em HL. A posição da pedra na tela é carregada em DE. Adicionando os conteúdos de DE e HL, obtemos a posição da pedra na TN. Porém, como estamos interessados no que existe abaixo da pedra, somamos 32 ao conteúdo de HL.

A rotina 74 é chamada e fornece o padrão da figura que se encontra nessa posição. Seu valor é comparado com 72 e 76, que são os códigos dos dois padrões de mar existentes. Se o mar está abaixo da pedra, a instrução **jr z,mo** manda o processador para a segunda parte da rotina, que apaga a pedra e ajusta a posição para o topo do morro.

POSIÇÃO

Uma vez que temos no acumulador o código do padrão que está abaixo da pedra, não custa verificar se ele corresponde ao céu. Para isso, comparamos o conteúdo do acumulador com 255, que é o código do padrão de céu.

Se não há céu abaixo da pedra — ou seja, se ela ainda está no chão — a instrução **jr nz,bo** faz o processador saltar ao rótulo **bo**, onde as variáveis são ajustadas antes de retornar. Caso contrário, o processador continua com a parte seguinte da rotina.

A PEDRA DESCE

Você deve ter reparado que até agora a pedra só tem se movido para a esquerda — o declive da encosta não foi levado em conta. A pedra só pode estar fora do chão (com a cor do céu sob ela) se acabou de passar por uma inclinação — o que significa que precisamos apagá-la e imprimi-la uma posição abaixo. A rapidez com que essa tarefa é executada em código de máquina nos dá a impressão de que a pedra permaneceu na superfície da encosta.

O endereço inicial da TN da VRAM é carregado em HL. O par DE recebe a posição atual da pedra e esse valor é somado em HL. Em seguida o código do padrão de céu é carregado em A e a rotina 77 da ROM é chamada. Com esse procedimento, a pedra é apagada da posição que ocupava na tela.

O valor em DE foi preservado na pilha e somado ao número 32 em HL, para atualizar a posição da pedra nos endereços -5200 e -5199 e levá-la uma posição abaixo da anterior. Finalmente, soma-se o endereço inicial da TN da VRAM em HL e carrega-se o acumulador com o padrão da pedra. A rotina 77 é, então, chamada. Com isso, fizemos a pedra descer uma posição, permanecendo firme junto à encosta.

ROLANDO PELA ENCOSTA

Quer a pedra tenha descido, quer não, as variáveis que definem sua posição devem ser ajustadas para fazer com que ela pareça estar rolando pela encosta da montanha.

O par HL é mais uma vez carregado com a posição da pedra. Em seguida, seu valor é decrementado e devolvido aos endereços -5200 e -5199. Quando a rotina responsável pela movimentação da pedra for chamada novamente, a figura terá sido deslocada uma posição para a esquerda.

O processador continuou a executar essa parte da rotina e imprimiu na tela a primeira figura da pedra porque o conteúdo do endereço -5195 era 0. Para que depois imprima a outra figura, o valor 1 será carregado em -5195 pelo acumulador.

SIMULAÇÃO E PREVISÃO

- FUNDAMENTOS E APLICAÇÕES
- SIMULAÇÃO DE UMA LOTERIA
- GERAÇÃO DE NÚMEROS RANDÔMICOS
- AMOSTRAGEM E PESQUISA

Quais são suas chances de fazer os treze pontos na loteria esportiva desta semana? Nosso programa não vai melhorar sua sorte, mas poderá ajudá-lo a entender porque perdeu.



Instruir um novo piloto a bordo de um avião a jato ou de um bombardeiro pode custar muito caro, se levarmos em conta os custos com combustível, aterrissagens e pessoal de apoio. Por isso, as autoridades militares preferem investir grandes somas em aviões de treinamento e simuladores — equipamentos controlados por computador que nunca deixam o solo, mas dão ao iniciante a sensação de um vôo real.

O mesmo tipo de argumento é válido em várias outras situações — incluindo o planejamento de uma indústria, marketing, pesquisas científicas e planos governamentais. Programar um computador para simular os efeitos de uma provável falência ou de uma determinada política econômica é, sem dúvida, mais razoável do que testá-las na prática, o que tomaria cinco anos ou mais, dependendo das circunstâncias. Não surpreendem, portanto, os grandes investimentos que têm sido feitos na área.

Antes do advento dos computadores, era praticamente impossível utilizar esse recurso, devido à complexidade e extensão dos cálculos envolvidos. Hoje em dia, até os microcomputadores pessoais podem fazer simulações não muito complicadas — como as que você já deve ter visto em jogos comercializados pelos fabricantes.

Por trás de todo trabalho de simulação e previsão, estão as leis da probabilidade. Se você não está familiarizado com elas, consulte o artigo *Acaso e Probabilidade* na página 776. Complementando essas leis, usam-se os estudos estatísticos, que analisam os fatos reais, quantificando a ocorrência de certas situações e o surgimento de variações em suas características.

A partir de algumas informações básicas, o computador é capaz de prever situações futuras para uma série de eventos. Mas a confiabilidade dos resul-

tados dependerá sempre do cuidado com que os dados foram introduzidos e da precisão das regras que orientam sua manipulação. Tomemos um exemplo bem simples: a simulação dos resultados de uma loteria esportiva.

A cada fim de semana, milhares de pessoas conferem, pela televisão ou pelo rádio, os resultados da loteria esportiva, aguardando ansiosamente sua vez de fazer os treze pontos. Infelizmente, todos, exceto os raros premiados, desligam a televisão desapontados e vão dormir, esperando pelos jogos da próxima semana. Se você quiser antecipar a emoção de checar sua sorte, digite estas linhas:

S

```
 10 DIM J(13)· BORDER 0: PAPER
 0: INK 7: CLS
 20 PRINT  INVERSE 1;" AS PART
 IDAS ESTAO EM ANDAMENTO"
 30 FOR I=1 TO 13
 40 LET J(I)=RND*1
 50 NEXT I
 60 FOR I=1 TO 2000: NEXT I
 70 PRINT : PRINT  FLASH 1;
 PAPER 2," E ATENCAO PARA OS R
 ESULTADOS·"  PRINT
 80 FOR I=1 TO 13
 90 IF J(I)<=.5 THEN  LET Z$="
 UM"
100 IF J(I)> 5 AND J(I)<=.75
THEN  LET Z$="DOIS"
110 IF J(I)>.75 THEN  LET Z$="
DO MEIO"
120 PRINT TAB 3;"JOGO ";I;:
PRINT TAB 11;": COLUNA ";Z$
130 FOR T=1 TO 500  NEXT T
140 NEXT I· PRINT · PRINT
150 PRINT  FLASH 0; PAPER 1;"V
OCE QUER OUTRA LOTERIA (S/N)?"
160 LET A$=INKEY$· IF A$=""
THEN  GOTO 160
170 IF A$="S" THEN  GOTO 10
180 STOP
```

T

```
10 DIM J(13)
20 CLS:PRINT " AS PARTIDAS ESTA
O EM ANDAMENTO"
30 FOR I=1 TO 13
40 LET J(I)=RND(10)/10
50 NEXT I
60 FOR I=1 TO 2000:NEXT
70 PRINT @34,"E ATENCAO PARA OS
 RESULTADOS":PRINT
80 FOR I=1 TO 13
90 IF J(I)<=.5 THEN Z$="UM"
100 IF J(I)>.5 AND J(I)<=.75 TH
EN Z$="DOIS"
110 IF J(I)>.75 THEN Z$="DO MEI
O"
120 PRINT TAB(4) "JOGO";I;": CO
LUNA ";Z$
130 FOR T=1 TO 500:NEXT T
140 NEXT I:PRINT
150 PRINT "VOCE QUER OUTRA LOTE
```



```
RIA ? (S/N)"
160 LET A$=INKEY$:IF A$="" THEN
 160
170 IF A$="S" THEN GOTO 20
180 END
```



```
10  DIM J(13)
20  HOME : PRINT  TAB( 5)"AS PA
RTIDAS ESTAO EM ANDAMENTO"
30  FOR I = 1 TO 13
40  LET J(I) =  RND (1)
50  NEXT I
60  FOR I = 1 TO 2000: NEXT
70  PRINT : PRINT  TAB( 6)"E AT
ENCAO PARA OS RESULTADOS": PRIN
T
80  FOR I = 1 TO 13
90  IF J(I) <  = .5 THEN Z$ = "
UM"
100  IF J(I) > .5 AND J(I) <  =
 .75 THEN Z$ = "DOIS"
110  IF J(I) > .75 THEN Z$ = "D
O MEIO"
120  PRINT  TAB( 7)"JOGO ";I;:
PRINT  TAB( 15)" ·    COLUNA ";Z
$
130  FOR T = 1 TO 500: NEXT T
140  NEXT I: PRINT : PRINT
150  PRINT  TAB( 4)"VOCE QUER O
UTRA LOTERIA ?(S/N)"
160  GET A$
170  IF A$ = "S" THEN  GOTO 20
180  END
```



```
10 DIMJ(13)
20 CLS:PRINT TAB(5)"AS PARTIDAS
 ESTÃO EM ANDAMENTO"
30 FOR I=1 TO 13
40 LET J(I)=RND(1)
50 NEXT I
60 FOR I=1 TO 2000:NEXT
70 PRINT:PRINT TAB(6) "E ATENÇA
O PARA OS RESULTADOS":PRINT
80 FOR I=1 TO 13
90 IF J(I)<=.5 THEN Z$="UM"
100 IF J(I)>.5 AND J(I)<=.75 TH
EN Z$="DOIS"
110 IF J(I)>.75 THEN Z$="DO MEI
O"
120 PRINT TAB(7) "JOGO";I; PRIN
T TAB(15)" ·    COLUNA ";Z$
130 FOR T=1 TO 500:NEXT T
140 NEXT I:PRINT:PRINT
150 PRINT TAB(4) "VOCE QUER OUT
RA LOTERIA? (S/N)"
160 A$=INKEY$:IF A$="" THEN GOT
O 160
170 IF A$="S" THEN GOTO 20
180 END
```

Rode o programa e os resultados serão prontamente impressos na tela

Não é incomum o cancelamento de algum jogo, sobretudo por causa de chuva. Quando isso ocorre, a Caixa Econômica Federal, a promotora da loteria, faz um sorteio para definir um resultado para o jogo. Na verdade, esse processo não passa de uma simulação,

ou seja, de uma representação simbólica de uma situação real.

A estrutura do programa é muito simples. O laço entre as linhas 30 e 50 sorteia treze números correspondentes aos treze jogos, e o laço entre as linhas 80 e 140 relaciona esses números aos possíveis resultados.

DECIDINDO OS RESULTADOS

O parâmetro que o computador utiliza para imprimir os dados do placar provém de uma análise sobre a frequência de cada resultado em jogos passados. Sabe-se que, em 50% dos jogos, o time da casa vence, pois conta, entre outras vantagens, com o apoio da torcida. As ocorrências restantes dividem-se igualmente entre empate — 25% — e vitória do time visitante — 25%.

Em cada jogo há três resultados possíveis (coluna um, coluna do meio e coluna dois) e a loteria inclui treze jogos. Assim, a probabilidade de ocorrer determinado resultado é de 1 em 1.594.323 (um em três elevado a treze). Como você vê, as chances de acerto seriam muito pequenas, caso não se permitissem apostas duplas ou triplas. Na Inglaterra, por exemplo, as apostas também incluem empate com gols ou sem gols, o que torna a probabilidade de ganhar ainda menor (1 em 4↑13).

Decidir o resultado de um jogo é, portanto, como fazer um sorteio: os números são colocados em uma caixa e alguém, sem olhar, pega um deles.

Suponhamos que se divida um papel em quatro partes: na primeira, escrevemos "coluna dois"; na segunda, "coluna do meio" e, nas outras duas, "coluna um". Se fizermos um sorteio, estaremos representando a decisão de uma partida de futebol. Também poderíamos escrever um número em cada pedaço de papel — 1, 2, 3 e 4 — e relacioná-los a um resultado desta forma:

NÚMERO	RESULTADO
1, 2	coluna um
3	coluna dois
4	coluna do meio

A função RND do BASIC, que gera números randômicos, faz com que o computador sorteie um número do "chapéu". Considerando que **RND** gera um número entre 0 e 1, podemos reescrever nossa tabela assim:

VALOR DO RND	RESULTADO
de 0 a 0.5	coluna um
maior que 0.5 até 0.75	coluna dois
maior que 0.75 até 1.00	coluna do meio

Voltando ao programa anterior, você pode observar que este é o processo utilizado pelo computador para decidir o resultado de cada jogo (linha 80 até linha 140).

Em nosso programa, as proporções entre os resultados dos jogos foram estimadas um pouco grosseiramente. Para aperfeiçoá-las, você poderá fazer uma pequena pesquisa em arquivos de jornais e redefinir a proporção a ser usada. Experimente fazer algumas previsões e confira com os resultados da próxima semana. Boa sorte!

GERAÇÃO DE NÚMEROS RANDÔMICOS

Dados, cartas e roleta já foram muito utilizados para a obtenção de números randômicos. Procedimentos como esses, lentos e tediosos, puderam ser abandonados depois que um famoso matemático americano, John von Newman, propôs o método da potência quadrada. Começando com um número de quatro dígitos (a "semente"), o próximo número randômico seria obtido pela multiplicação da semente por ela mesma. Do resultado seriam destacados os quatro dígitos do meio. Suponhamos que a semente é o número 5272. O segundo número será gerado pelos quatro dígitos centrais de (5272↑2), ou seja, 27.793.984. A resposta (7939) é praticamente randômica. Um terceiro número seria obtido elevando-se 7939 ao quadrado e assim por diante.

Você pode estar se perguntando se qualquer processo matemático — obrigatoriamente repetitivo — seria capaz de gerar números realmente randômicos. Na verdade, isso é impossível. Porém, o número obtido comporta-se como se fosse randômico, e é geralmente chamado pseudo-randômico.

Infelizmente, a técnica da potência quadrada não é muito útil para gerar números randômicos no computador. Além de ser um processo lento, a seqüência logo se repete, e, assim que surge um zero, ela é interrompida. A maioria dos micros adota um método distinto, recorrendo a uma fonte qualquer de números para gerar os pseudo-randômicos. O próximo programa usa uma fórmula bem simples e a função **INT** para demonstrar como esse método funciona.

S

```
20 BORDER 0: INK 7: PAPER 0:
CLS
30 PRINT AT 0,2; INVERSE 1;"
NUMEROS PSEUDO-RANDOMICOS "''
40 INPUT " QUANTOS NUMEROS ?
";n: LET s=0
```





Quantas variáveis cabem na memória?

Nos interpretadores Microsoft BASIC dos micros MSX, TRS-80, TRS-Color, Apple e TK-2000, os nomes de variáveis podem ter uma ou duas letras, ou uma letra e um número. Também é possível acrescentar o sufixo de tipo, que pode ser um # (precisão dupla), ! (precisão simples), % (inteiro) e $ (literal). Dadas essas limitações, calcula-se o número de nomes diferentes em 2 000.

Já para os micros Sinclair (ZX-81 e Spectrum), não há limite teórico, pois são aceitos mais de dois caracteres no nome de uma variável.

```
50 LET  x =.677829*PEEK 23673
/50
60 LET x=x*1842.95
70 LET x=x-INT (x)
80 LET p=INT (x*1000): PRINT
"      ";.001*p,
90 LET s=s+1
110 IF s<=n THEN  GOTO 60
120 STOP
```



```
20 CLS
30 PRINT @3,"NUMEROS PSEUDO-RAN
DOMICOS"
40 PRINT:PRINT:INPUT"QUANTOS NU
MEROS ";N:S=0
50 X=.677829*TIMER/50
60 X=X*1842.95
70 X=X-INT(X)
80 P=INT(X*1000):PRINT LEFT$(ST
R$(P/1000)+"          ",8);
90 S=S+1
110 IF S<=N THEN 60
120 END
```



```
20  HOME
30  PRINT  TAB( 7)"NUMEROS PSEU
DO-RANDOMICOS": PRINT
40  INPUT "QUANTOS NUMEROS ? ";
N:S = 0
50 X = .677829 * 318378 / 50
60 X = X * 1842.95
70 X = X -  INT (X)
80 P =  INT (X * 1000): PRINT
LEFT$ ( STR$ (P / 1000) + "
   ",8);
90 S = S + 1
110  IF S < N THEN 60
120  END
```



```
20 CLS
30 PRINT TAB(6) "NUMEROS PSEUDO
-RANDOMICOS":PRINT
40 INPUT"QUANTOS NUMEROS ";N:S=
0
50 X=.677829*TIME/50
60 X=X*1842.95
70 X=X-INT(X)
80 P=INT(X*1000):PRINT LEFT$(ST
R$(P/1000)+"          ",8);
90 S=S+1
110 IF S<N THEN 60
120 END
```

A linha 50 usa a função *time* para produzir uma semente diferente a cada execução do programa. O Apple e o TK-2000 não possuem essa função. Portanto, se você quiser uma outra sequência, deverá substituir o valor 318378.

O número .677829 é uma constante arbitrária. Depois que o valor é multiplicado por outra constante (linha 60), obtém-se o resíduo (ou parte decimal). Mude os valores das constantes nas linhas 50 e 60 e torne a executar o programa para verificar o tipo de resultado a que chegará.

Para outros fins, é mais fácil utilizar a função **RND** existente em seu computador. Variando o valor de **x** na expressão **RND(x)**, você poderá selecionar uma sequência que se repita, o que é muito útil para renovar a semente a cada execução do programa.

Lembre-se de que a função **RND** cria uma variável randômica e não uma variável algébrica. **RND(1)** — ou **RND(0)** no TRS-Color — não resultará no mesmo número em duas partes do programa.

AMOSTRAS E PESQUISAS

Empresas que fazem pesquisas de mercado ou de opinião pública usam computadores para gerar uma amostra randômica da população. Quando entrevistam, por exemplo, mil pessoas, é muito importante que elas representem a grande maioria da população. Ou seja, os entrevistados não podem ser escolhidos por morar perto do instituto de pesquisa ou trabalhar em determinada empresa. Ao contrário, devem ter as mais diversas origens e características para que suas opiniões não sejam influenciadas por particularidades comuns a determinados grupos.

A melhor maneira de se definir uma amostra bem representativa é selecioná-la randomicamente, excluindo, assim, qualquer tendência ou preconceito.

Isso continua valendo mesmo que o campo amostral não seja tão grande, e que comporte de fato algumas características comuns — como se supõe haver entre os sócios de um clube ou os leitores de INPUT. Se você quisesse saber, por exemplo, quais os computadores mais usados entre os leitores de INPUT, seria necessário fazer uma amostragem randômica para chegar ao resultado mais próximo possível do real.

O processo utilizado no programa é similar ao de retirar números de um chapéu, com uma pequena diferença: o papel sorteado não voltará ao chapéu, o que equivaleria a entrevistar duas vezes a mesma pessoa. Execute o programa que se segue para ver como a função **RND** pode ser empregada para gerar uma amostragem randômica.



```
 10 DIM b$(10,16)
 20 DIM a$(10,16)
 30 BORDER 0: PAPER 0: INK 7:
 CLS
 50 PRINT AT 0,5; INVERSE 1;"A
 MOSTRAGEM RANDOMICA "
 90 PAUSE 100: CLS
100 FOR i=1 TO 10: READ a$(i):
 NEXT i
110 INPUT " TAMANHO DA AMOSTRA
 ? ",n
120 FOR v=1 TO 10: LET b$(v)=a
$(v): NEXT v
130 FOR j=1 TO n
140 LET r=1+INT (RND*10)
150 IF b$(r)="
" THEN  GOTO 140
160 PRINT b$(r)
170 LET b$(r)=""
180 NEXT j
190 INPUT " OUTRA AMOSTRA (s/n
) ? ";g$
200 IF g$="s" THEN  CLS : GOTO
 110
210 STOP
220 DATA "BONN","COPENHAGUE","
LONDRES"
230 DATA "MADRI","MOSCOU","NOV
A IORQUE"
240 DATA "PARIS","ROMA","ESTOC
OLMO","VIENA"
```



```
40 CLS
50 PRINT @8,"AMOSTRA RANDOMICA"
:PRINT:PRINT
100 RESTORE:FOR I=1 TO 10:READ
A$(I):NEXT I
110 INPUT"TAMANHO DA AMOSTRA ";
N:PRINT
115 IF N>10 THEN 40
120 FOR V=1 TO 10:B$(V)=A$(V):N
EXT V
130 FOR J=1 TO N
140 R=1+INT(RND(0)*10)
150 IF B$(R)="" THEN 140
160 PRINT B$(R)
170 B$(R)=""
180 NEXT J
190 PRINT:INPUT" OUTRA AMOSTRA
(S/N) ";G$:PRINT:PRINT
200 IF G$="S" THEN 110
210 END
220 DATA BONN,COPENHAGUE,LONDRE
```

```
S
230 DATA MADRI,MOSCOU,NOVA IORQ
UE
240 DATA PARIS,ROMA,ESTOCOLMO,V
IENA
```



```
40  HOME
50  PRINT  TAB( 10)"AMOSTRAGEM
RANDOMICA": PRINT
100  RESTORE : FOR I = 1 TO 10:
 READ A$(I): NEXT I
110  INPUT "TAMANHO DO CAMPO AM
OSTRAL :";N: PRINT
115  IF N > 10 THEN 40
120  FOR V = 1 TO 10:B$(V) = A$
(V): NEXT V
130  FOR J = 1 TO N
140 R = 1 +  INT ( RND (1) * 10
)
150  IF B$(R) = "" THEN 140
160  PRINT B$(R)
170 B$(R) = ""
180  NEXT J
190  PRINT : INPUT "OUTRA AMOST
RAGEM ?(S/N) ";G$: PRINT : PRIN
T
200  IF G$ = "S" THEN 40
210  END
220  DATA  BONN,COPENHAGEN,LOND
RES
230  DATA  MADRI,MOSCOU,NOVA YO
RK
240  DATA   PARIS,ROMA,ESTOCOLM
O,VIENA
```



```
20 CLS
30 PRINT TAB(6) "NUMEROS PSEUDO
-RANDOMICOS":PRINT
40 CLS
50 PRINT TAB(8) "AMOSTRAGEM RAN
DOMICA":PRINT PRINT
100 RESTORE:FOR I=1 TO 10:READ
A$(I):NEXT I
110 INPUT "TAMANHO DA AMOSTRA "
;N:PRINT
120 FOR V=1 TO 10 B$(V)=A$(V):N
EXT V
130 FOR J=1 TO N
140 R=1 +INT(RND(1)*10)
150 IF B$(R)="" THEN 140
160 PRINT B$(R)
170 B$(R)=""
180 NEXT J
190 PRINT INPUT"OUTRA AMOSTRA ?
 (S/N)";G$:PRINT:PRINT
200 IF G$="S" THEN 110
210 END
220 DATA BONN,COPENHAGEN,LONDRE
S
230 DATA MADRI,MOSCOU,NOVA YORK
240 DATA PARIS,ROMA,ESTOCOLMO,V
IENA
```

Depois que as informações que seguem o comando **DATA** foram lidas (linha 100), um número inteiro randômico **R**, entre 1 e 10, é gerado (linha 140). Seu computador imprimirá, então, o **R**-ésimo item (linha 160) da lista.

Assim que um item é selecionado, deve-se removê-lo do banco de informações, para não ser escolhido duas vezes. A linha 170 se encarrega disso.

Geralmente, o banco de informações a ser pesquisado é bem mais extenso do que o apresentado aqui. Para obter amostragens de um grupo maior, nosso programa seria lento e ineficiente. Se um político quisesse, por exemplo, sortear duzentos nomes de um eleitorado de 60.000 elementos, o programa teria que percorrer a lista duzentas vezes. Para evitar essa longa espera, o mais conveniente seria recorrer ao método de busca individual.

O MÉTODO DE BUSCA INDIVIDUAL

Com esse método, o banco de dados é percorrido uma só vez, de cima para baixo. Decide-se, a cada nome considerado, qual deve ser incluído na amostragem. Para obter uma amostragem dessa maneira, faça as seguintes modificações no programa anterior:



```
120 LET a=n: LET c=10
130 FOR j=1 TO 10
140 IF a=0 THEN  GOTO 190
150 IF RND*1<=a/c THEN  PRINT
a$(j): GOTO 170
160 LET c=c-1: GOTO 180
170 LET a=a-1: LET c=c-1
```



```
50 PRINT @3,"AMOSTRA RANDOMICA
SIMPLES ":PRINT:PRINT
120 A=N:C=10
130 FOR J=1 TO 10
140 IF A=0 THEN 190
150 IF RND(0)<=A/C THEN PRINT A
$(J):GOTO 170
160 C=C-1:GOTO 180
170 A=A-1:C=C-1
```



```
50  PRINT "AMOSTRAGEM RANDOMICA
 DE BUSCA INDIVIDUAL": PRINT
120 A = N:C = 10
130  FOR J = 1 TO 10
140  IF A = 0 THEN 190
150  IF  RND (1) <  = A / C THE
N  PRINT A$(J): GOTO 170
160 C = C - 1: GOTO 180
170 A = A - 1:C = C - 1
```



```
50 PRINT TAB(9) "AMOSTRAGEM RAN
DOMICA":PRINT:PRINT TAB(9) "DE
BUSCA INDIVIDUAL":PRINT
120 A=N:C=10
130 FOR J=1 TO 10
140 IF A=0 THEN 190
150 IF RND(1)<=A/C THEN PRINT A
$(J):GOTO 170
160 C=C-1 GOTO 180
170 A=A-1:C=C-1
```

MICRO DICAS

FAZENDO PREVISÕES

As técnicas de simulação descritas neste artigo apresentam muitos dos elementos necessários para que você próprio desenvolva um método para prever os resultados de um jogo ou para fazer uma pesquisa. Não será difícil modificar os programas ou usar parte deles, adaptando-os às suas finalidades.

Você pode, por exemplo, planejar e montar um programa que contenha o resultado de vinte partidas e, então, selecionar treze delas e comparar suas previsões com os resultados reais. Para isso, seria preciso utilizar tanto o programa da loteria esportiva quanto o da amostragem, com algumas modificações e linhas extras destinadas a ligá-los.

Se você quiser treinar um pouco mais, experimente uma outra combinação dos dois programas, fazendo com que o computador não só simule os resultados da loteria esportiva, como também forneça os nomes dos times que estão jogando. Você deverá, nesse caso, incluir no programa de amostragem os nomes de vários times, dentre os quais 26 serão sorteados.

Se você decidir selecionar três itens da lista, o primeiro deles, Bonn, só será escolhido se a função **RND** (linha 150) resultar num número menor que 3/10. Copenhague será o próximo item a ser considerado. Se Bonn já tiver entrado para a amostragem, o item Copenhague terá a chance de ser escolhido se a função **RND** gerar um valor menor que 2/9. Se Bonn não foi selecionado, a chance de Copenhague aumenta para 3/9. As linhas 160 e 170 atualizam essas probabilidades para cada elemento.

Se você comparar as amostragens obtidas por este programa com as do programa anterior, observará que, aqui, elas aparecem na ordem em que estão no programa.

À primeira vista, pode-se supor que, com uma lista de dez itens, o número de amostragens possíveis é pequeno. Na verdade, porém, é bem grande: são 120 amostras diferentes com três itens, e 252 com cinco itens.

AVALANCHE: AS PEDRAS ROLAM (2)

Willie precisará ter muito cuidado agora. As pedras estão prontas para rolar morro abaixo. E, nesta segunda parte da rotina, descerão numa avalanche sobre ele.

Até agora Willie não contava com a ameaça de uma avalanche. A rotina que faz com que as pedras rolem está apenas pela metade e, provavelmente, falhará se você tentar executá-la. Esta segunda parte da rotina irá precipitar a avalanche. E Willie só estará seguro se pular fora do caminho.

A listagem que apresentamos aqui é constituída de duas seções principais. Uma delas imprime a segunda figura da pedra — para dar a impressão de que ela está rolando — e verifica se Willie foi atingido. A outra apaga a pedra se ela tiver chegado ao fim da encosta ou à superfície do mar e a recoloca no topo da encosta. Ambas as seções são chamadas pela parte da rotina fornecida no artigo anterior.

Assim que você tiver digitado e montado as linhas que se seguem, a rotina de movimentação da pedra estará pronta para funcionar. Mas lembre-se de que você deve ter o resto do jogo na memória, já que outras rotinas — como **print** — serão chamadas.

```
 10 REM org 59097
 20 REM bma ld hl,(57356)
 30 REM ld de,22528
 40 REM add hl,de
 50  REM ld a,(hl)
 60 REM cp 40
 70 REM jr nz,bnh
 80 REM ld a,2
 90 REM ld (57336),a
100 REM bnh ld hl,(57356)
110 REM ld a,42
120 REM ld bc,57128
130 REM call 58217
140 REM inc hl
150 REM call 58217
160 REM ld a,0
170 REM ld (57358),a
180 REM ret
190 REM bri ld hl,(57356)
200 REM ld bc,15616
210 REM ld a,45
220 REM call 58217
230 REM ld hl,223
240 REM ld (57356),hl
250 REM ret
```

Na última parte de *Avalanche*, você teve que analisar a cor do caractere que estava logo abaixo da pedra, para verificar se ela tinha chegado à água ou se estava rolando. Agora, será preciso checar a cor da posição onde a pedra será impressa, para saber se ela conseguirá atingir Willie.

A posição de impressão da pedra é transferida dos endereços 57356 e 57357 para o par HL. Não se esqueça de que a variável de posição da pedra, que está nesses endereços, foi decrementada no fim da primeira parte desta rotina. Assim, quando a rotina é novamente chamada, a variável está apontando para a posição à esquerda da pedra. Saltando para essa seção da rotina, o programa imprime a segunda figura da pedra no local indicado — uma posição à esquerda da anterior —, para dar a impressão de que está em movimento.

O par DE é carregado com 22528, valor adicionado em seguida ao conteúdo de HL. O resultado, que aponta para a cor da posição onde a pedra será impressa, permanece no par HL.

Utilizamos o apontador em HL para obter a cor da posição e carregá-la no acumulador por meio de endereçamento direto. Essa cor é comparada com 40, que corresponde ao código de azul sobre fundo azul ciano, a cor de Willie contra o céu.

Se a cor nessa posição não é 40, a pedra não atingirá Willie, e a instrução **jr nz,bnh** faz o processador pular as próximas instruções.

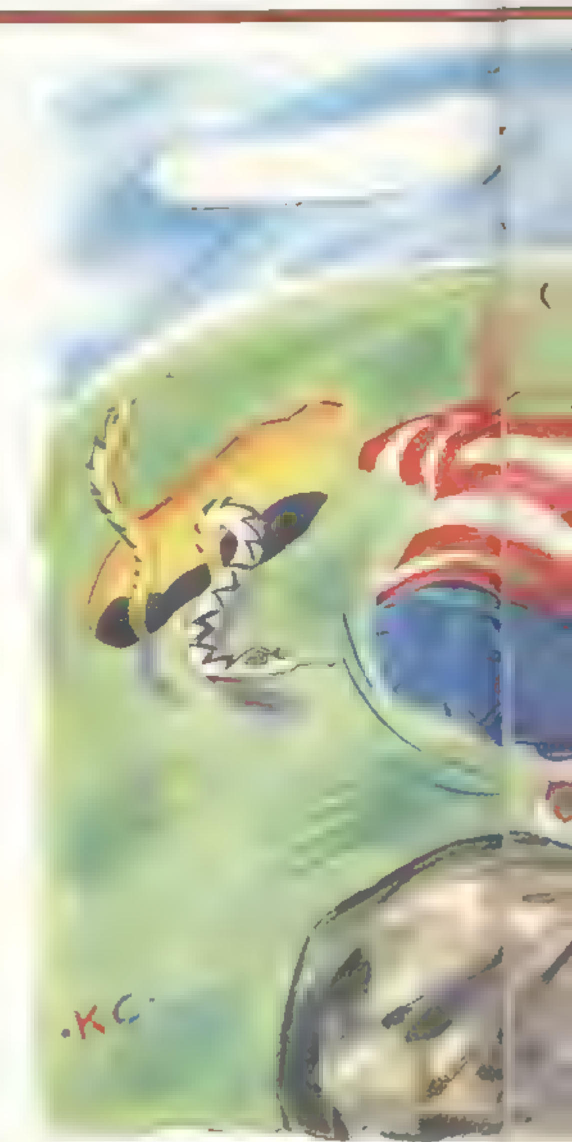
WILLIE É ATINGIDO

Se Willie for mortalmente atingido pela pedra, a variável no endereço 57336 deve ser igualada a 2. Para isso, colocamos 2 no acumulador e carregamos seu conteúdo em 57336.

A PEDRA ESTÁ ROLANDO

Inevitavelmente, a pedra irá rolar uma posição à esquerda. Se Willie não estiver pelo caminho, nem se preocupe. Caso contrário, você pode ter a certeza de que ele foi esmagado.

O par HL é carregado com a nova posição da pedra na tela. O acumulador



A é carregado com 42 — vermelho sobre fundo ciano — e o par BC, com 57128, que corresponde ao endereço inicial dos dados para a segunda figura da pedra, que ocupa duas posições na tela. A rotina **print** é chamada e imprime a primeira metade da pedra (a porção esquerda). O par HL é incrementado — o que faz o apontador se mover uma posição para a direita. A rotina **print** é novamente chamada, imprimindo a segunda metade da pedra.

Como você verá, a segunda pedra se

- IMPRESSÃO DA SEGUNDA PEDRA
- EFEITO DE ANIMAÇÃO
- WILLIE FOI ATINGIDO?
- MORTE DO INFELIZ

PERSONAGEM

- FIM DA ENCOSTA
- MERGULHO NO MAR
- DE VOLTA AO TOPO
- A AVALANCHE RECOMEÇA



desloca meio caractere de cada vez, pois ocupa duas posições na tela. Essa estrutura de duas metades faz com que o movimento seja muito mais suave e contínuo, acentuando a impressão de que a pedra está rolando.

Note que, movendo a pedra um caractere para a esquerda (como fizemos na primeira parte da rotina) e, depois, meio caractere, mantemos um avanço tão suave na posição em 57356 e 57357 que não há necessidade de verificar se a pedra ainda está no chão.

Agora, basta que igualemos a variável do tipo de pedra a 0, para que, quando a rotina for chamada de novo, o processador execute a outra parte. Para isso, colocamos 0 no acumulador e carregamos o conteúdo de A em 57358.

RECOMEÇA A AVALANCHE

A rotina **bri** é chamada pela primeira parte da rotina de movimentação sempre que a pedra chega ao fim da encosta ou afunda nas águas do mar. A função de **bri** consiste em apagar a pedra da sua posição atual e reajustá-la para o topo da encosta.

A operação de apagar a pedra é feita como de costume. Sua posição é transferida de 57356 e 57357 para HL; o par BC é carregado com o endereço inicial dos dados (que estão na ROM) para um espaço vazio e o acumulador A, com o valor 45, que corresponde ao código de ciano sobre fundo ciano. Chamada em seguida, a rotina **print** imprime um caractere cor do céu sobre a pedra, fazendo-a desaparecer da tela.

O par HL é carregado com 223, posição da pedra na tela quando ela se encontra no topo da montanha. Esse valor é colocado de volta nos endereços 57356 e 57357, para que a rotina de movimentação da pedra comece nessa posição, quando for novamente chamada.

A rotina aqui apresentada tem duas seções principais. Ambas são chamadas pela parte do programa publicada no artigo anterior da série *Avalanche*. A primeira delas, que começa no rótulo **BOK**, seleciona e imprime na tela um dos dois padrões de pedra. Em seguida, troca o valor da variável que controla esses padrões, para que a outra figura seja impressa na próxima vez.

A segunda seção, que começa no rótulo **BRI**, apaga a pedra se ela tiver chegado ao fim da encosta ou à água, recolocando-a no topo do morro.

Assim que tiver digitado e montado as linhas que se seguem, a rotina de movimentação da pedra estará pronta para funcionar. Mas lembre-se de que você deve ter o resto do jogo na memória, já que outras rotinas, como **CHARPR**, por exemplo, serão chamadas.

```
10   ORG 19853
20   BOK LDA 18260
30   BEQ BMN
40   LDX 18253
50   LDU #18038
60   JSR CHARPR
70   CLR 18260
80   RTS
90   BMN LDX 18253
```

```
100  LDU #18014
110  JSR CHARPR
120  LDA #1
130  STA 18260
140  RTS
150  BRI LDX 18253
160  LDU #1536
170  JSR CHARPR
180  LDX #3070
190  STX 18253
200  RTS
210  CHARPR EQU 19402
```

Existem, na memória, dois padrões diferentes da pedra, que são impressos na tela alternadamente, dando a impressão de que a figura esta rolando

Para a criação desse efeito, o processador precisa saber qual das duas figuras foi impressa na última vez. Obtém tal informação consultando uma baliza no endereço 18260. O conteúdo dessa posição é carregado no acumulador. Se for 0, a instrução **BEQ BMI** salta para a rotina que imprime uma das figuras da pedra. Se não for, o programa continua e imprime a outra.



A PEDRA ROLA

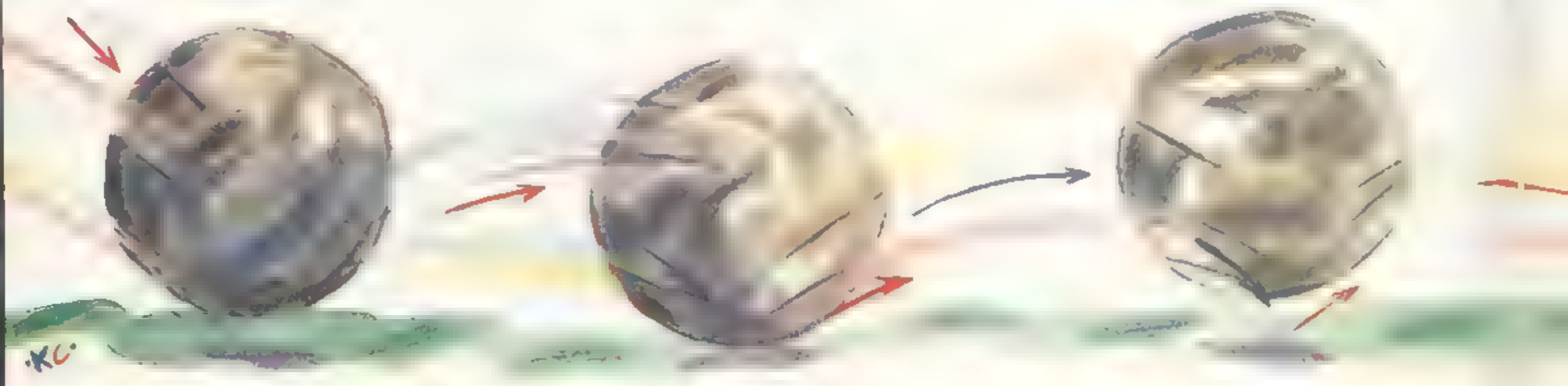
Caso o programa continue, X é carregado com o conteúdo da posição de memória 18253. Esse endereço armazena a posição onde a pedra irá ser impressa. O registrador U é carregado com o número 18038, endereço inicial de uma das figuras da pedra.

O processador salta para a rotina **CHARPR**, que imprime os últimos oito bytes da pilha do usuário na posição de tela que está sendo apontada pelo conteúdo do registrador X.

A instrução **CLR 18260** altera a baliza no endereço 18260. O processador a executa quando a baliza tem valor 1. **CLR 18260** apaga esse valor, colocando 0 em seu lugar. Assim, na próxima vez que a rotina da pedra for chamada, o processador se encarregará de executar a outra parte e imprimir a segunda figura da pedra. Depois, a baliza é zerada e o processador retorna.

A parte seguinte dessa rotina simplesmente imprime a outra figura da pedra. O registro é carregado com o mesmo valor, mas o apontador da pilha do usuário, U, é carregado com o endereço inicial na memória da outra figura da pedra, 18104. Em seguida, a rotina **CHARPR** é chamada e realiza a impressão na tela. O processador executou essa parte da rotina porque a baliza em 18260 tinha o valor 0. Conseqüentemente, 1 é carregado no acumulador e armazenado em 18260. A troca da baliza faz com que, na próxima vez, a outra figura da pedra seja impressa.

A rotina **BRI** é chamada quando a pedra chegou ao fim da encosta ou atingiu a água. Sua função consiste em apagar a pedra e inicializar sua posição para o topo da montanha. A instrução





LDX 18253 carrega no registrador X a posição atual da pedra na tela; U é carregado com o endereço inicial de um caractere de céu na memória. A rotina **CHARPR** imprime, então, um bloco de céu sobre a pedra, apagando-a.

O registrador X é carregado com 3070, posição inicial da pedra na tela quando ela se encontra no topo da montanha. Esse valor é armazenado em 18253, a variável que carrega a posição em que a pedra será impressa na próxima vez.

Para testar a rotina de movimentação da pedra, execute estas linhas em BASIC com o resto do programa.

```
5 POKE 30000,57
10 EXEC 19426
20 EXEC 19781
30 FOR K=1 TO 100.NEXT:GOTO 20
```

A linha 5 só será necessária se você tiver feito a gravação da rotina da música separadamente.



A rotina que apresentamos a seguir tem duas seções principais. Ambas são chamadas pela parte do programa publicada no artigo anterior da série *Avalanche*. A primeira seção imprime a segunda figura da pedra, que se alternará com a primeira, dando a impressão de que a pedra está rolando. Além disso, verifica também se nosso personagem foi atingido pela pedra.

A segunda seção apaga a pedra caso ela tenha chegado ao final da encosta ou à superfície do mar e inicializa sua posição no topo da montanha.

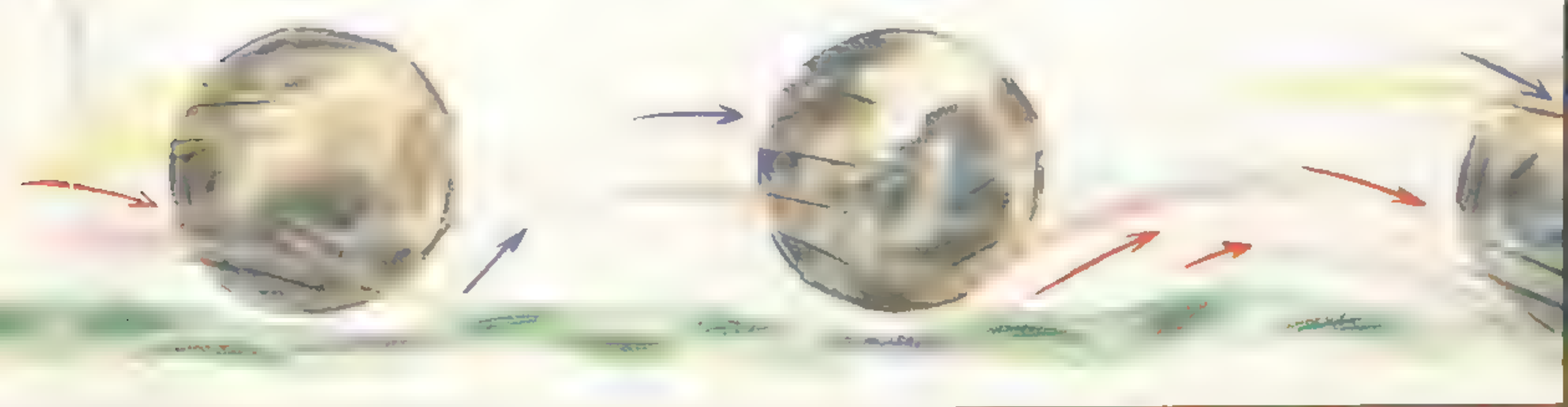
Depois de digitar e montar as próximas linhas, a rotina de movimentação da pedra estará completa e pronta para funcionar. Mas lembre-se de que o resto do jogo precisa estar na memória, pois as tabelas de padrões e de cores são necessárias nesta rotina. Para obter o efeito desejado, é preciso, também, que a montanha esteja na tela.

```
10    org 54530
20    ld hl,(62407)
30    ld de,(-5200)
40    add hl,de
50    push hl
60    call 74
70    cp 1
80    jr z,bn
90    cp 5
100    jr z,bn
110    cp 7
120    jr z,bn
130    cp 11
140    jr nz,bu
150    bn ld a,2
160    ld (-5201),a
170    bu pop hl
180    ld a,17
190    push hl
200    call 77
210    pop hl
220    inc hl
230    ld a,19
240    call 77
250    ld a,0
260    ld (-5195),a
270    ret
280    org -10953
290    ld hl,(62407)
300    ld de,(-5200)
310    add hl,de
320    ld a,255
330    call 77
340    ld hl,255
350    ld (-5200),hl
360    ret
370    end
```

Na primeira parte da rotina de movimentação, você teve que examinar o padrão que estava logo abaixo da pedra, para verificar se ela tinha afundado no mar ou se estava rolando. Identificaremos agora o padrão que está na posição onde a pedra será impressa, para saber se ela atingirá ou não Willie.

A posição de impressão da pedra, que está armazenada em −5200 e −5199, é colocada no par DE. Lembre-se de que a variável de posição da pedra, que estava nesses endereços, foi decrementada no fim da primeira parte desta rotina. Assim, quando a rotina é novamente chamada, a variável está apontando para a posição à esquerda da pedra. Ao executar essa parte da rotina, o processador imprime a segunda figura da pedra, dando a impressão de que ela está rolando.

O par HL é carregado com o endereço inicial da Tabela de Nomes da VRAM (TN). A posição que está em DE é somada ao endereço em HL, que passa a conter efetivamente o endereço equiva-



MICRO DICAS

LEITURA E ESCRITA NA VRAM DO MSX

A VRAM é tratada pelo MSX como se fosse um periférico. Portanto, são necessários alguns artifícios para colocarmos dados nessa memória auxiliar, especialmente em linguagem de máquina, quando não dispomos dos comandos **VPEEK** e **VPOKE**.

Em nosso jogo, usamos cinco rotinas para escrever e ler na VRAM. Seus endereços são: 74, 77, 86, 89 e 92. Todas empregam os comandos **OUT** e **IN** para escrever e ler através da porta 152. As rotinas dos endereços 80 e 83 controlam o destino do byte enviado por essa porta.

lente a essa posição na tela. Em seguida, o valor de HL é guardado na pilha, sendo utilizado na impressão da pedra.

A rotina 74 da ROM é chamada. Ela coloca no acumulador o valor da posição da VRAM apontada por HL, ou seja, realiza a leitura da VRAM. O acumulador passa o conter o código do padrão que está à esquerda da pedra. Esse código é comparado com 1, 5, 7 e 11, valores que correspondem aos diferentes desenhos das pernas de Willie. Caso o valor lido não seja nenhum desses, a instrução **jr nz,bu** faz o processador pular as próximas instruções.

WILLIE É ATINGIDO

Se o pobre Willie tiver sido mortalmente atingido pela pedra, a variável no endereço −5201 deve ser ajustada com o valor 2. Para isso, colocamos 2 no acumulador e carregamos o conteúdo deste em −5201.

A PEDRA ROLA

Inevitavelmente a pedra irá rolar uma posição à esquerda. Se Willie não estiver nessa posição, não se preocupe. Caso contrário, podemos ter a certeza de que ele foi esmagado.

A posição na Tabela de Nomes da VRAM é recuperada da pilha, voltando para o par de registros HL. O acumulador A é carregado com o código do primeiro padrão da figura da pedra (a metade esquerda será impressa primeiro). O valor de HL é novamente carregado na pilha, pois será usado na impressão da outra metade. A rotina 77 da ROM é chamada. Ela coloca o valor contido no acumulador na posição na VRAM apontada por HL. Já a utilizamos várias vezes na série *Avalanche*.

O endereço da VRAM é recuperado da pilha para HL, onde é incrementado. Esse par de registros passa, então, a apontar para uma posição à direita na tela, onde será impressa a metade que está faltando. O acumulador é carregado com 19, o código dessa metade, e a rotina 77 é chamada outra vez.

A segunda figura da pedra ocupa duas posições na tela, movendo-se meio caractere de cada vez. Essa estrutura de duas metades faz com que o movimento se realize de modo muito mais suave e contínuo, acentuando a impressão de que a pedra está rolando.

O deslocamento da posição da pedra, um caractere para a esquerda (como ocorreu na primeira parte da rotina) e, depois, meio caractere, mantém uma variação tão suave na posição em −5200 e −5199, que não há necessidade de verificar se a pedra está no chão.

Agora, falta apenas um detalhe: ajustar a variável do tipo de pedra a 0, para que, quando a rotina for novamente chamada, o processador execute a outra parte. Para isso, colocamos 0 no acumulador e carregamos seu conteudo no endereço −5195.



Por que não usamos sprites para representar as pedras no MSX?

Os sprites poderiam, de fato, apresentar algumas vantagens sobre os blocos gráficos. Com eles, obteríamos os movimentos mais suaves e também não precisaríamos apagar a última posição da figura já que isso é feito automaticamente.

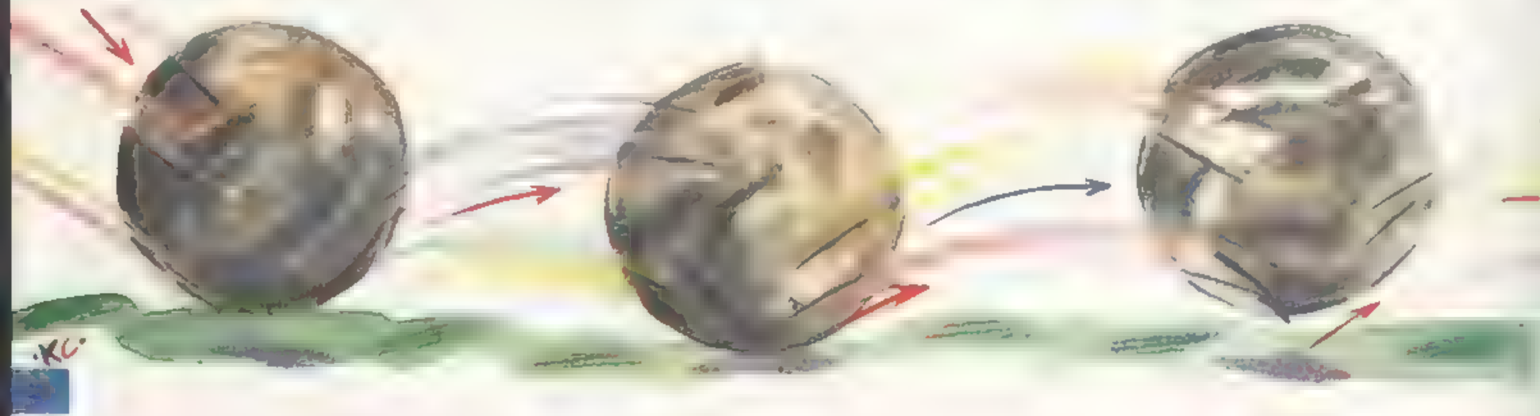
Mas não foi sem razão que optamos pelos blocos gráficos. Primeiro, só é permitida uma cor no sprite. Além disso, a detecção da colisão de um sprite com outras estruturas complicaria muito o programa. Por fim, não poderíamos ter, como requer o jogo, mais do que quatro figuras simultaneamente na mesma linha.

RECOMEÇA A AVALANCHE

A rotina **mo** é chamada pela primeira parte da rotina de movimentação sempre que a pedra chega ao fim da encosta ou atinge o mar. Ela apaga a pedra da sua posição atual e reajusta essa posição para o topo da montanha.

Para apagar a pedra, o endereço inicial da TN da VRAM é colocado no par de registros HL. A esse endereço soma-se a posição atual da pedra, que está em DE. O valor 255, que corresponde ao código do padrão de céu, é colocado no acumulador. A rotina 77 é chamada e imprime o padrão de céu na posição que a pedra ocupava.

O par HL é carregado com 255, a posição da pedra no topo da encosta. Esse valor volta para −5200 e −5199. Quando a rotina de movimentação for chamada, a pedra estará nessa posição.



DISCOS RÍGIDOS

Embora muito úteis, os disquetes têm um inconveniente: a baixa capacidade de armazenamento. Se você precisa de maior espaço de memória e velocidade de acesso, o disco rígido é a solução.

Em artigos anteriores, discutimos as características e as vantagens dos discos flexíveis (também chamados *floppies* ou disquetes) para os usuários de microcomputadores. Esses periféricos são formidáveis, em termos de capacidade de armazenamento, facilidade de uso e velocidade de acesso, quando comparados com outras formas de gravação magnética de informação.

Entretanto, também apresentam desvantagens. A principal é a incapacidade de atender a demandas de armazenamento maiores do que as habituais.

Os disquetes de face simples (para as linhas TRS-80, TRS-Color, Apple e TK 2000) têm capacidade em torno dos 160-180 Kbytes. Pode parecer muito, mas é suficiente apenas para cerca de noventa páginas de texto, ou alguns programas e arquivos de dados pequenos. Os disquetes de dupla face têm capacidade de armazenamento de cerca de 320-360 Kbytes — o que também não é muito, considerando-se a espantosa rapidez com que um usuário médio enche até centenas de disquetes.

Não seria interessante ter todos os programas e arquivos de dados em um único disco? Para isso, existe uma solução: o disco rígido. Embora ainda seja um periférico muito caro (sobretudo no Brasil), seu preço tende a se tornar acessível. Não há exagero em afirmar que, mais cedo ou mais tarde, todos os micros pessoais serão vendidos com uma unidade embutida de disco rígido, como já ocorre com os micros profissionais da linha PC-XT.

O QUE É UM DISCO RÍGIDO

O disco rígido (*hard disk*, em inglês) é feito de metal, e não de plástico flexível, como o disquete. Por essa razão, apresenta mais estabilidade térmica e estrutural, o que lhe permite maiores velocidades de rotação (e acesso) e maior densidade de gravação. Essas características resultam numa grande capacidade de armazenamento.

A desvantagem do disco rígido é que, em geral, ele não pode ser removido e trocado facilmente por outro, como o disquete, pois é fixo dentro da unidade acionadora. Existem unidades de disco rígido que são totalmente intercambiáveis — inclusive a cabeça de gravação e leitura.

Nos discos rígidos mais utilizados para micros, a cabeça de gravação e leitura nunca entra em contato com a superfície do disco, como acontece com o disquete. Ela "sobrevoa" a superfície a uma distância muito pequena (a alguns milésimos de milímetro); por isso, o desgaste da superfície ferromagnética é praticamente nulo.

Esse sistema garante maior durabilidade à unidade, mas requer que ela seja isolada do exterior, por meio de vácuo ou fluxo forçado de ar. A partícula mais ínfima de poeira ou fumaça que se introduzir entre a cabeça e o disco pode danificá-lo.

Essa tecnologia é conhecida como *Winchester*, denominação que se costuma estender aos próprios discos rígidos. Como ela envolve dispositivos mecânicos e eletrônicos complexos e delicados, os custos dos discos rígidos tornam-se bem mais caros.

CAPACIDADE E VELOCIDADE

A capacidade de armazenamento dos discos Winchester é espantosa. Os de menor capacidade têm por volta de 5 Mbytes de espaço (5 milhões de caracteres), o que equivale a cerca de trinta disquetes de face simples!

São cada vez mais populares os discos rígidos de 10, 15 e 20 Mbytes, embora as maiores capacidades possam ser usadas apenas por micros de 16 e 32 bits. Mas já existem discos de boa capacidade para o Apple e o TRS-80.

A velocidade do disco rígido é cerca de vinte a trinta vezes maior que a dos disquetes, dependendo do modelo. Isso faz uma enorme diferença no momento de carregar um programa extenso na memória do micro.

COMO CONECTAR UM DISCO RÍGIDO

Desde que existam modelos de disco rígido disponíveis para o seu microcomputador, você não terá dificuldades em utilizar este periférico.

Em primeiro lugar, será necessário uma interface controladora — uma caixinha ou placa, que pode ser conectada ao computador (internamente, por exemplo, nos micros da linha Apple), facultando-lhe o controle e intercâmbio de dados com a unidade de disco.

Muitos micros de oito bits têm placas de controle para disquetes que já incluem o controlador de discos rígidos. Em alguns casos, porém, é preciso adquirir uma interface própria.

A unidade de disco rígido é apresentada normalmente em duas versões: em gabinetes separados do console do computador, e em "gavetas" que podem ser inseridas no local destinado a uma unidade acionadora de disquetes. Em alguns computadores (como os PC) é possível instalar a unidade Winchester em espaço reservado no gabinete da UCP, sem tomar o lugar de um disquete.

Não convém dispor apenas do disco rígido: é melhor ter também uma ou duas unidades de disquetes — só assim poderemos obter a cópia cautelar (*back-up*) do disco rígido e a transferência de programas e dados.

O disco rígido tem de fato a desvantagem de requerer freqüentemente cópias de seu conteúdo em disquete, pois, se ocorrer um defeito, a perda é total. Por isso, muitos usuários adquirem junto com o Winchester uma unidade de fita de back-up (chamada *streamer*), que possibilita a execução dessa cópia de uma vez só, e em poucos minutos.

Resta considerar a fonte de alimentação. O disco rígido gasta mais energia do que uma unidade de disquetes, exigindo, quase sempre, uma fonte de alimentação separada da UCP. Alguns fabricantes vendem a unidade de disco rígido com fonte própria de alimentação, o qual é muito conveniente, sobretudo se a unidade não é interna.

UMA PLANILHA ELETRÔNICA (2)

Descubra para onde vai o seu dinheiro ou planeje o futuro de seus negócios utilizando esta prática planilha. Adicione mais uma parte ao programa iniciado no artigo anterior.

Quando uma planilha começa a ser planejada e está como uma folha em branco, muitas vezes ainda não sabemos exatamente de que maneira iremos aproveitá-la. Os exemplos dados no artigo anterior e as sugestões que aqui apresentamos vão ajudá-lo a definir uma planilha que seja realmente útil. O programa permite a montagem de diversas planilhas, que você poderá gravar e recarregar a qualquer momento.

Uma planilha para registrar e planejar suas despesas domésticas, onde as entradas aparecem sob títulos como aluguel, transporte, saúde, consertos etc. constitui uma boa opção. Mas, se você tem que fazer um número muito grande de consertos ou reformas na casa, por exemplo, pode ser interessante montar uma planilha só para eles. Nesse caso, separe os diferentes tipos de reforma — estofamento dos móveis, decoração, troca das telhas e calhas, colocação de grades nas janelas —, especificando as quantias gastas mensalmente ou trimestralmente com cada um. O programa se encarregará de fornecer os valores totais para cada categoria, assim como seu peso relativo no conjunto dos gastos com reformas.

Uma outra folha pode incluir as despesas da família com itens como alimentação, vestuário, educação, saúde, transporte e lazer. Utilize-a para listar esses gastos por semanas, meses ou por pessoas da família.

Mas lembre-se de que pode recorrer à planilha para manipular qualquer tipo de informação que necessite de uma organização lógica. Uma planilha para sócios de um clube pode conter, por exemplo, os nomes, telefones e mensalidades pagas, assim como os comparecimentos a reuniões.

Qualquer que seja sua escolha, a planilha se revelará um excelente instrumento de controle. Se você consultar o artigo da página 201, verá que ela nada mais é que uma sofisticada matriz bidimensional. Permite um controle maior dos dados porque comporta o uso de fórmulas, o que torna possível a obtenção imediata dos resultados.

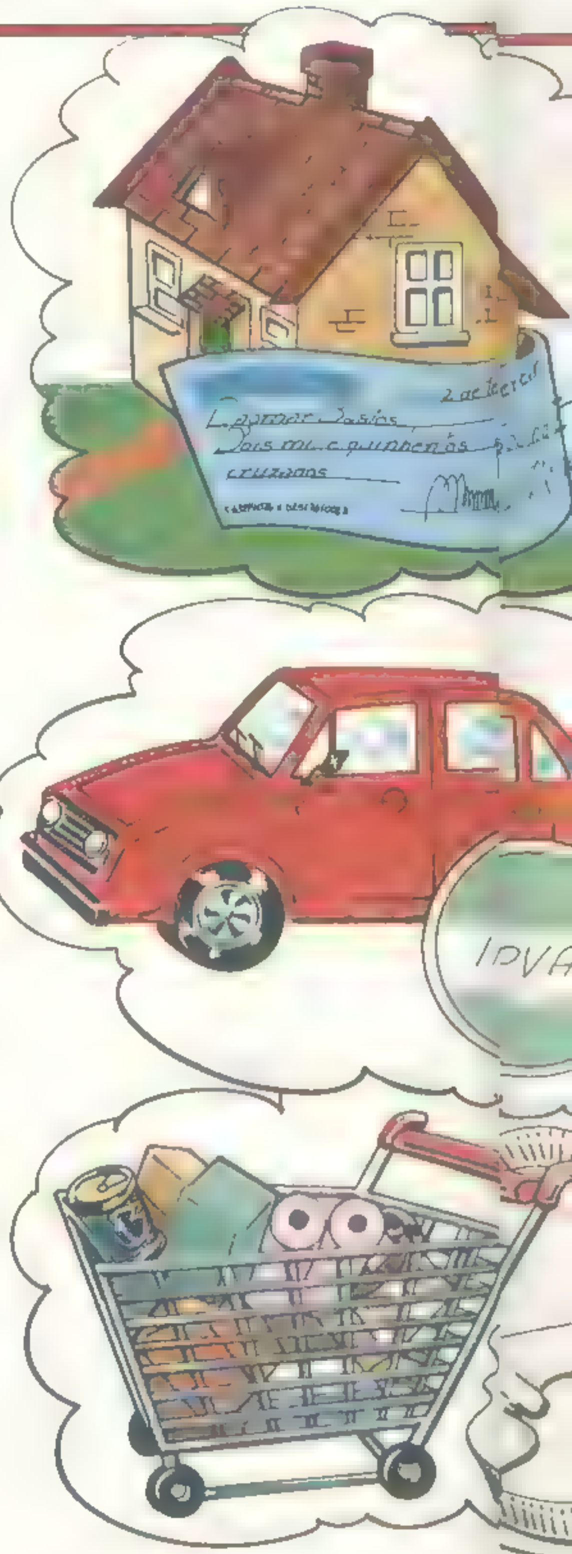
Aplicações financeiras constituem o tipo mais freqüente de uso. Mas as variações, nessa área, também são ilimitadas. Planilhas podem conter detalhes de pedidos, descrições de itens, evolução nos custos, descontos etc. São úteis, também, na elaboração de folhas de pagamento, fornecendo listagens dos nomes dos empregados e calculando horas trabalhadas, salários e adiantamentos. Sua cooperação é, ainda, valiosa no controle de estoque e contas, em geral; no planejamento de orçamentos de firmas e em muitos outras tarefas ligadas ao setor empresarial.

DIGITE O PROGRAMA

A parte do programa aqui listada deve ser adicionada à que apresentamos no artigo anterior. As linhas restantes serão dadas no último artigo da série, que conterá instruções detalhadas sobre o uso do programa. Assim, carregue a listagem anterior, digite esta e grave o programa para, depois, acrescentar a terceira parte.

S

```
420 FOR a=fc TO c: FOR b=fr TO
tr
430 IF z$(3,2)="C" THEN  LET v
(2)=v(2)+1: LET v(4)=v(4)+(v(3
)<>26)
440 IF z$(3,2)="R" THEN  LET v
(3)=v(3)+(v(3)<>26): LET v(1)=
v(1)+1
450 IF v(1)<25 AND v(2)<31 AND
v(3)=26 THEN  GOTO 470
460 IF v(1)>24 OR v(2)>30 OR v
(3)>24 OR v(4)>30 THEN  GOTO
 570
470 IF v(1)<1 OR v(2)<1 OR v(3
)<1 OR v(4)<1 THEN  GOTO 570
480 LET a$=CHR$ (v(1)+64)+STR$
v(2)+CHR$ (v(3)+64)+STR$ v(4)+
o$
490 LET c=LEN a$: IF c>8 THEN
RETURN : RESTORE 1630: FOR q=1
TO 11: LET f=0: READ m$: FOR w
=1 TO c
500 IF m$(w)="A" THEN  GOSUB
1650: IF f THEN  GOTO 560
510 IF m$(w)="N" THEN  GOSUB
1670: IF f THEN  GOTO 560
520 IF m$(w)="Z" THEN  GOSUB
1710: IF f THEN  GOTO 560
530 IF m$(w)="O" THEN  GOSUB
1690: IF f THEN  GOTO 560
```





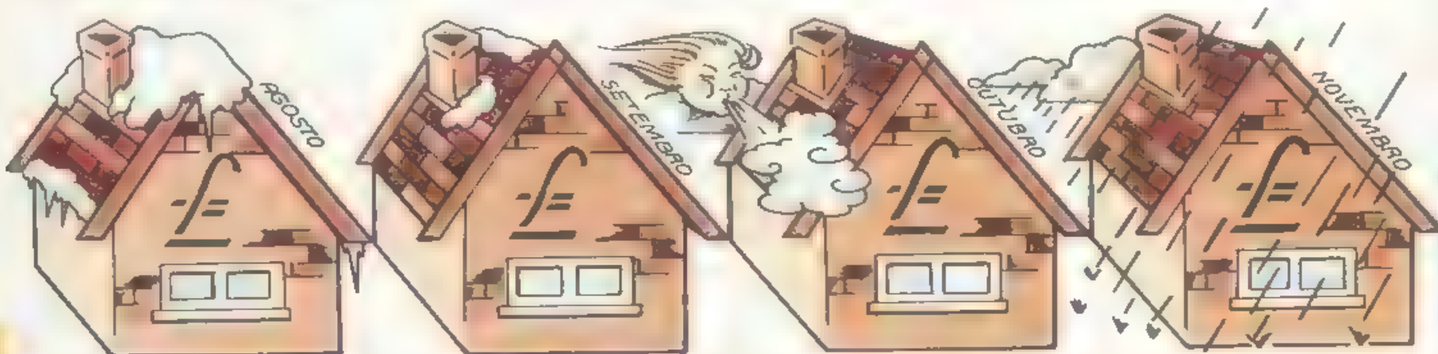
VIAGEM
IPVA
CHEQUES

```
540 NEXT w: LET z=q: GOSUB
1140: IF NOT f THEN LET s$="
      ": FOR w=1 TO c: LET s$
(w)=a$(w): NEXT w: LET d$(b,a,
9 TO 16)=s$: LET d$(b,a,18)=
CHR$ z: LET d$(b,a,17)="1":
NEXT b: NEXT a: RETURN
550 GOTO 570
560 NEXT q
570 RETURN
580 LET e=c: LET a$="    "
590 PRINT #1;AT 0,x; BRIGHT 1;
" "
600 PAUSE 0: LET i=CODE INKEY$
610 IF i>88 THEN GOTO 600
620 IF i=13 THEN GOTO 650
630 IF i=12 THEN LET a$(4-e)=
" ": LET e=e+1: LET x=x-1
640 LET a$(4-e)=CHR$ i: PRINT
#1;AT 0,x;CHR$ i: LET x=x+1:
LET e=e-1: IF e>0 THEN PAUSE
10: GOTO 590
650 IF e>1 AND (d=1 OR d=4 OR
d=5) THEN GOTO 590
655 IF e>0 AND (d=2 OR d=3)
THEN GOTO 590
660 PAUSE 10: PRINT #1;AT 0,0;
"
  ": RETURN
670 LET i$=""
680 FOR z=1 TO 3
690 LET i$=i$+(a$(z) AND a$(z)
<>" ")
700 NEXT z
710 IF LEN i$=3 THEN IF i$(1)
<"A" OR i$(1)>"X" OR i$(2)<"0"
OR i$(2)>"9" OR i$(3)<"0" OR i
$(3)>"9" THEN LET f=1: RETURN

720 IF LEN i$=3 THEN IF VAL
i$(2 TO 3)=0 OR VAL i$(2 TO 3)
>30 THEN LET f=1: RETURN
730 IF LEN i$=2 THEN IF i$(1)
<"A" OR i$(1)>"X" OR i$(2)<"1"
OR i$(2)>"9" THEN LET f=1:
RETURN
740 IF d=2 THEN IF i$(1)<>"A"
AND i$(1)<>"R" THEN LET f=1:
RETURN
750 IF d=3 THEN IF i$(1)<>"C"
AND i$(1)<>"R" THEN LET f=1:
RETURN
760 LET z$(d,2 TO )=i$: LET z$
(d,1)=CHR$ (LEN i$+48): LET f=
0: RETURN
770 LET fc=(CODE z$(4,2))-64:
LET tc=(CODE z$(5,2))-64: LET
fr=VAL z$(4,3 TO (1+VAL z$(4,1
))): LET tr=VAL z$(5,3 TO (1+
VAL z$(5,1)))
780 IF z$(3,2)="C" THEN IF fc
<>tc OR fr>tr THEN LET f=1:
RETURN
790 IF z$(3,2)="R" THEN IF fc
>tc OR fr<>tr THEN LET f=1:
RETURN
800 LET f=0: RETURN
810 FOR y=1 TO 30: FOR x=1 TO
24: LET os=0
820 IF d$(y,x,17)="1" THEN
LET z=CODE d$(y,x,): GOSUB 880
: GOSUB 1010: LET st=1: LET a$
=STR$ t: LET os=LEN a$: IF t>
99999.99 THEN LET f$(y,x,1)=
"5"
830 IF os>8 THEN LET st=os-7
840 IF os=0 THEN GOTO 860
850 LET s$="        ": FOR u=s
t TO os: LET s$(u-st+1)=a$(u):
NEXT u: GOSUB 1410: LET d$(y,x
, TO 8)=s$
860 LET i=IN 32766: IF i=252
THEN PRINT #1;AT 0,0; PAPER 2
; INK 7;"CALCULO ABANDONADO":
RETURN
870 NEXT x: NEXT y: RETURN
880 LET s$=d$(y,x,9 TO 16)
890 IF z=1 THEN LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2): LET v(3)=(CODE s$(3))-64
: LET v(4)=VAL s$(4): LET o$=s
$(5): RETURN
900 IF z=2 THEN LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2 TO 3): LET v(3)=(CODE s$(4
)-64): LET v(4)=VAL s$(5): LET
 o$=s$(6): RETURN
```



```
660 A$=D$(I,J):B$=MID$(A$,2)
670 AT=ASC(A$)
680 IF AT=128 THEN PRINT STRING
$(7,32);:GOTO 720
690 IF AT=129 OR AT=130 THEN PR
INT USING "$     $";B$;:GOTO 72
0
700 FOR U=1 TO LEN(B$):IF MID$(
B$,U,1)<>CHR$(32) THEN PRINT MI
D$(B$,U,1);
710 NEXT U
720 RETURN
730 C1=ASC(Z$)-64:C2=VAL(MID$(Z
$,2)):V=D(C1,C2):RETURN
740 PRINT @448,"TRABALHANDO"
750 FOR J=1 TO RX
760 FOR I=1 TO CX
770 D(I,J)=0:IF ASC(D$(I,J))=12
9 THEN D(I,J)=VAL(MID$(D$(I,J),
2))
780 NEXT I,J
790 FOR J=1 TO RX
800 FOR I=1 TO CX
810 PRINT @448,"TRABALHANDO NA
CELULA ",CHR$(I+64),MID$(STR$(J
),2)
820 IF ASC(D$(I,J))<>131 THEN 1
130
830 A$=MID$(D$(I,J),2)
840 O$=MID$(A$,7,1)
850 IF O$="&" THEN 1050
860 IF O$="$" THEN 1090
870 Z$=LEFT$(A$,3)
880 GOSUB 730
890 V1=V:Z$=MID$(A$,4,3):GOSUB
730:V2=V
900 DP=VAL(RIGHT$(A$,1))
910 ON INSTR (1,OP$,O$) GOSUB 1
000,1010,1020,1030,1040
920 OV=0:IF DP=0 THEN PU$="####
###":MP=7:GOTO 950
930 PU$=STRING$(7-(DP+1),"#")+"
."+STRING$(DP,"#"):MP=7-(DP+1)
940 IF LEN(PU$)>7 THEN RV$="  <
OV>":OV=1
950 D(I,J)=RV
960 IF RV<0 THEN MP=MP-1
970 ML=LEN(MID$(STR$(INT(RV+.5)
),2))
980 IF ML>MP THEN RV$="  <OV>":
OV=1
990 GOTO 1160
1000 RV=V1+V1:RETURN
1010 RV=V1-V2:RETURN
1020 RV=V1*V2:RETURN
1030 IF V2=0 THEN RV=0:RETURN E
LSE RV=V1/V2:RETURN
1040 RV=V1*V2/100:RETURN
1050 P1=ASC(A$)-64:P2=ASC(MID$(
A$,4,1))-64:C2=VAL(MID$(A$,2,2)
):RV=0
1060 FOR C1=P1 TO P2
1070 RV=RV+D(C1,C2):NEXT
1080 DP=VAL(RIGHT$(A$,1)):GOTO
920
1090 P1=VAL(MID$(A$,2,2)):P2=VA
L(MID$(A$,5,2)):C1=ASC(A$)-64:R
V=0
1100 FOR C2=P1 TO P2
1110 RV=RV+D(C1,C2):NEXT
1120 DP=VAL(RIGHT$(A$,1)):GOTO
920
1130 IF ASC(D$(I,J))<>128 THEN
```



```
1150
1140 RV$=STRING$(7,32):GOTO 116
0
1150 RV$=MID$(D$(I,J),2)
1160 IF I>=CS AND I<=CS+3 AND J
>=RS AND J<=RS+11 THEN PRINT @(
J-RS)*32+35+(I-CS)*7,"";:PF=1 E
LSE PF=0
1170 IF(ASC(D$(I,J))>=128 AND A
SC(D$(I,J))<=130) OR OV=1 THEN
1200
1180 IF PF=1 THEN PRINT USING P
U$,RV,
1190 GOTO 1210
1200 IF PF=1 THEN PRINT USING"%
     %";RV$;
1210 NEXT I,J
1220 RETURN
1230 CLS:INPUT"DESEJA SALVAR ES
TA FOLHA (S/N) ";A$
1240 IF A$<>"S" THEN 1340
1250 LINE INPUT"NOME DO ARQUIVO
: ";F$
1260 OPEN "O",# 1,F$
1270 FOR J=1 TO RX
1280 FOR I=1 TO CX
1290 IF ASC(D$(I,J))=128 THEN 1
320
1300 Z$=D$(I,J):MID$(Z$,1,1)=CH
R$(ASC(MID$(Z$,1,1))-95)
1310 PRINT # 1,STR$(I),STR$(J):
PRINT # 1,Z$
1320 NEXT I,J
1330 CLOSE # 1
1340 CLS:MO=1:GOSUB 70:RETURN
1350 CLS:PRINT"DESEJA CARREGAR
UMA FOLHA DO     GRAVADOR ?":PRI
NT"(O CONTEUDO DA MEMORIA SERA
    ADICIONADO AO DA FITA)":INP
UT "S / N ";A$
1360 IF A$<>"S" THEN 1480
1370 PRINT"PRESSIONE <ENTER> PA
RA CARREGAR O PROXIMO ARQUIVO D
A FITA  OU   DIGITE NOME DO ARQ
UIVO DESEJADO":PRINT
1380 LINE INPUT"NOME DO ARQUIVO
: ",F$
1390 OPEN "I",#-1,F$
1400 IF EOF(-1) THEN 1470
1410 INPUT#-1,A$,B$:LINE INPUT#
-1,C$
1420 MID$(C$,1,1)=CHR$(ASC(MID$
(C$,1,1))+95
1430 C1=VAL(A$):C2=VAL(B$):D$(C
1,C2)=C$
1440 IF C1>CX THEN CX=C1
1450 IF C2>RX THEN RX=C2
1460 GOTO 1400
1470 CLOSE #-1
1480 CLS:CC=1:CR=1:CS=1:RS=1:MO
=1:GOSUB 70:RETURN
```



```
660 A$=D$(I,J):B$=MID$(A$,2)
670 AT=ASC(A$)
680 IF AT=128 THEN PRINTSTRING$
(7,32);:GOTO 720
690 IF AT=129 OR AT=130 THEN PR
INTUSING"\      \";B$;:GOTO 720
700 FOR U=1 TO LEN(B$):IF MID$(
B$,U,1)<>CHR$(32)THEN PRINTMID$
(B$,U,1);
710 NEXTU
720 RETURN
730 C1=ASC(Z$)-64:C2=VAL(MID$(Z
$,2)):V=D(C1,C2):RETURN
740 LOCATE 0,20:PRINT"TRABALHAN
DO...";SPC(24)
750 FOR J=1 TO RX
760 FOR I=1 TO CX
770 D(I,J)=0:IF ASC(D$(I,J))=12
9 THEN D(I,J)=VAL(MID$(D$(I,J),
2))
780 NEXT I,J
790 FOR J=1 TO RX
800 FOR I=1 TO CX
810 LOCATE 0,20:PRINT"TRABALHAN
DO NA CEL ";CHR$(I+64);MID$(STR
$(J),2)
820 IF ASC(D$(I,J))<>131 THEN 1
130
830 A$=MID$(D$(I,J),2)
840 O$=MID$(A$,7,1)
850 IF O$="&" THEN 1050
860 IF O$="$" THEN 1090
870 Z$=LEFT$(A$,3)
880 GOSUB 730
890 V1=V:Z$=MID$(A$,4,3):GOSUB
730:V2=V
900 DP=VAL(RIGHT$(A$,1))
910 ON INSTR(1,OP$,O$) GOSUB 10
00, 1010, 1020, 1030, 1040
920 OV=0:IF DP=0 THEN PU$="####
###":MP=7:GOTO 950
930 PU$=STRING$(7-(DP+1),"#")+"
."+STRING$(DP,"#"):MP=7-(DP+1)
940 IF LEN(PU$)>7 THEN RV$="  <
OV>":OV=1
950 D(I,J)=RV
960 IF RV<0 THEN MP=MP-1
970 ML=LEN(MID$(STR$(INT(RV+.5)
),2))
980 IF ML>MP THEN RV$="  <OV>":
OV=1
990 GOTO 1160
1000 RV=V1+V2:RETURN
1010 RV=V1-V2:RETURN
1020 RV=V1*V2:RETURN
1030 IF V2=0 THEN RV=0:RETURN E
LSE RV=V1/V2:RETURN
1040 RV=V1*V2/100:RETURN
1050 P1=ASC(A$)-64:P2=ASC(MID$(
A$,2,2)):RV=0
1060 FOR C1=P1 TO P2
1070 RV=RV+D(C1,C2):NEXT
1080 DP=VAL(RIGHT$(A$,1)):GOTO
920
1090 P1=VAL(MID$(A$,2,2)):P2=VA
L(MID$(A$,5,2)):C1=ASC(A$)-64:R
V=0
1100 FOR C2=P1 TO P2
1110 RV=RV+D(C1,C2):NEXT
1120 DP=VAL(RIGHT$(A$,1)):GOTO
920
1130 IF ASC(D$(I,J))<>128 THEN
1150
1140 RV$=STRING$(7,32):GOTO 116
0
1150 RV$=MID$(D$(I,J),2)
1160 IF I>=CS AND I<=CS+4 AND J
>=RS AND J<=RS+15 THEN LOCATE (
I-CS)*7+3,(J-RS)+1:PF=1 ELSE PF
=0
1170 IF (ASC(D$(I,J))>=128 AND
ASC(D$(I,J))<=130) OR OV=1 THEN
 1200
1180 IF PF=1 THEN PRINTUSINGPU$
;RV;
1190 GOTO 1210
1200 IF PF=1 THEN PRINTUSING"\
   \";RV$;
1210 NEXT I,J
1220 RETURN
1230 CLS:INPUT"QUER GRAVAR ESTA
 FOLHA? (S/N) ";A$
1240 IF A$<>"S" THEN 1340
1250 LINEINPUT "NOME DO ARQUIVO
: ";F$
1260 F$="CAS:"+F$:OPEN F$ FOR O
UTPUT AS #1
1270 FOR J=1 TO RX
1280 FOR I=1 TO CX
1290 IF ASC(D$(I,J))=128 THEN 1
320
1300 Z$=D$(I,J):MID$(Z$,1,1)=CH
R$(ASC(MID$(Z$,1,1))-90)
1310 PRINT#1,STR$(I);",";STR$(J
):PRINT#1,Z$
```



```
1320 NEXT I,J
1330 CLOSE #1
1340 CLS.MO=1·GOSUB 70:RETURN
1350 CLS·INPUT"QUER CARREGAR UM
A FOLHA DO TAPE?         (NOTE QU
E A FOLHA DA MEMORIA SERA COMBI
 NADA COM A NOVA) ";A$
1360 IF A$<>"S" THEN 1480
1380 LINEINPUT "NOME DO ARQUIVO
: ";F$
1390 F$="CAS·"+F$:OPEN F$ FOR I
NPUT AS #1
1400 IF EOF(1) THEN 1470
1410 INPUT #1.A$.B$:LINEINPUT #
1,C$
1420 MID$(C$,1,1)=CHR$(ASC(MID$
(C$,1,1))+90)
1430 C1=VAL(A$):C2=VAL(B$):D$(C
1,C2)=C$
1440 IF C1>CX THEN CX=C1
1450 IF C2>RX THEN RX=C2
1460 GOTO 1400
1470 CLOSE #1
1480 CLS·CC=1:CR=1:CS=1:RS=1:MO
=1·GOSUB70·RETURN
```

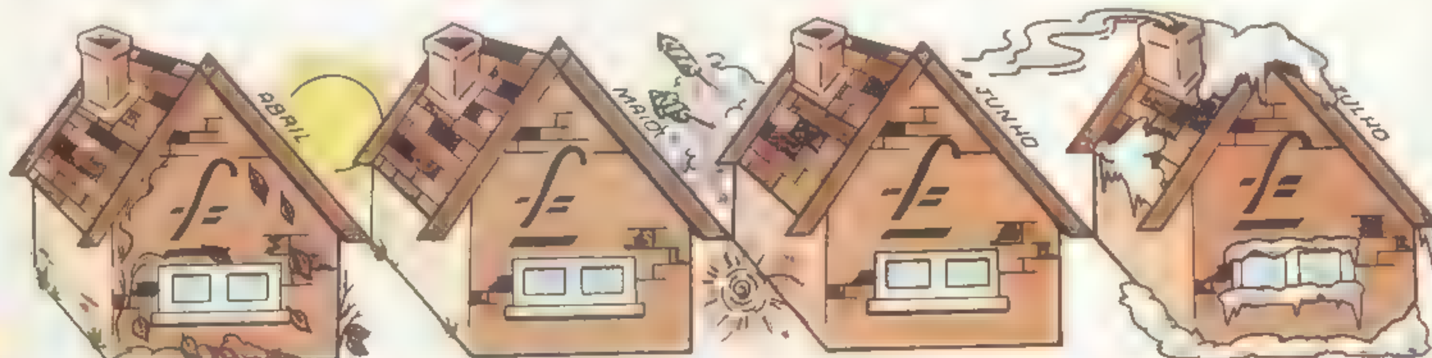


```
660 A$ = D$(I,J)·B$ =  MID$ (A$
,2)
670 AU =  ASC (A$)
680  IF AU = 128 THEN  PRINT  S
PC( 7);: GOTO 720
690  IF AU = 129 OR AU = 130 TH
EN  PRINT  LEFT$ (B$,7); SPC( 7
 -  LEN ( LEFT$ (B$,7))): GOTO
720
700 UU = 0: FOR U = 1 TO  LEN (
B$)  IF  MID$ (B$,U,1) <  >  CH
R$ (32) THEN  PRINT  MID$ (B$,U
,1);·UU = UU + 1
710  NEXT · PRINT  SPC( 7 - UU)

720  RETURN
730 C1 =  ASC (Z$) - 64:C2 =  V
AL ( MID$ (Z$,2))·V = D(C1,C2):
 RETURN
740  VTAB 21  HTAB 1· PRINT "TR
ABALHANDO"
750  FOR J = 1 TO RX
760  FOR I = 1 TO CX
770 D(I,J) = 0: IF  ASC (D$(I,J
)) = 129 THEN D(I,J) =  VAL ( M
ID$ (D$(I,J),2))
780  NEXT · NEXT
790  FOR J = 1 TO RX
800  FOR I = 1 TO CX
810  VTAB 21: HTAB 1: PRINT "TR
ABALHANDO NA CEL "; CHR$ (I + 6
4),J
820  IF  ASC (D$(I,J)) <  > 131
 THEN 1130
830 A$ =  MID$ (D$(I,J),2)
840 O$ =  MID$ (A$,7,1)
850  IF O$ = "&" THEN 1050
860  IF O$ = "$" THEN 1090
870 Z$     LEFT$ (A$,3)
880  GOSUB 730
890 V1 = V:Z$ =  MID$ (A$,4,3):
 GOSUB 730·V2 = V
900 DP =  VAL ( RIGHT$ (A$,1))
905 IN = 1
910  IF O$ <  >  MID$ (OP$,IN,1
) THEN IN = IN + 1· GOTO 910
915  ON IN GOSUB 1000,1010,1020
,1030,1040
920 OV = 0:MP = 7
950 D(I,J) = RV
960  IF RV < 0 THEN MP = MP - 1

970 ML =  LEN ( STR$ ( INT (RV
+ .5)))
980  IF ML > MP THEN RV$ = "  <
OV>"·OV = 1
990  GOTO 1160
1000 RV = V1 + V2: RETURN
1010 RV = V1 - V2: RETURN
1020 RV = V1 * V2: RETURN
1030  IF V2 = 0 THEN RV = 0· RE
TURN
1035 RV = V1 / V2· RETURN
1040 RV = V1 * V2 / 100· RETURN

1050 P1 =  ASC (A$) - 64:P2 =
ASC ( MID$ (A$,4,1)) - 64 C2 =
 VAL ( MID$ (A$,2,2))·RV = 0
1060  FOR C1 = P1 TO P2
1070 RV = RV + D(C1,C2): NEXT
1080 DP =  VAL ( RIGHT$ (A$,1))
: GOTO 920
1090 P1 =  VAL ( MID$ (A$,2,2))
·P2 =  VAL ( MID$ (A$,5,2))·C1
=  ASC (A$) - 64:RV = 0
1100  FOR C2 = P1 TO P2
1110 RV = RV + D(C1,C2): NEXT
1120 DP =  VAL ( RIGHT$ (A$,1))
: GOTO 920
1130  IF  ASC (D$(I,J)) <  > 12
8 THEN 1150
1140 RV$ = "       "
1150 RV$ =  MID$ (D$(I,J),2)
1160  IF I >  = CS AND I <  = C
S + 4 AND J >  = RS AND J <  =
RS + 11 THEN  VTAB J - RS + 2·
HTAB (I - CS) * 7 + 4:PF = 1: G
OTO 1170
1165 PF = 0
1170  IF ( ASC (D$(I,J)) >    1
28 AND  ASC (D$(I,J)) <  = 130)
 OR OV = 1 THEN 1200
1180  IF PF = 1 THEN  PRINT  SP
C( 7 -  LEN ( LEFT$ ( STR$ (RV)
,7))); LEFT$ ( STR$ (RV),7),
1190  GOTO 1210
1200  IF PF = 1 THEN  PRINT RV$
,
1210  NEXT : NEXT
1220  RETURN
1230  HOME   INPUT "QUER GRAVAR
 ESTA FOLHA? (S/N) ".A$
1240  IF A$ <  > "S" THEN 1340
1250  INPUT "NOME DO ARQUIVO ".
F$
1260  PRINT D$;"OPEN ";F$: PRIN
T D$;"WRITE ";F$
1270  FOR J = 1 TO RX
1280  FOR I = 1 TO CX
1290  IF  ASC (D$(I,J), = 128 T
HEN 1320
1300 Z$ = D$(I,J):Z$ =  CHR$ (
ASC (Z$) - 90) +  MID$ (Z$,2)
1310  PRINT  STR$ (I), CHR$ (13
); STR$ (J); CHR$ (13).Z$
1320  NEXT : NEXT
1330  PRINT D$;"CLOSE"
1340  HOME ·MO = 1· GOSUB 70  R
ETURN
1350  HOME : INPUT "QUER CARREG
AR UMA FOLHA DO DISCO?        (N
OTE QUE A FOLHA DA MEMORIA SERA
 COMBI NADA COM A NOVA)     (S/N
)".A$
1360  IF A$ <  > "S" THEN 1480
1380  INPUT "NOME DO ARQUIVO ";
F$
1390  PRINT D$;"OPEN ";F$
1400  PRINT D$;"READ ";F$
1410  INPUT A$.B$.C$
1420 C$ =  CHR$ ( ASC (C$) + 90
) +  MID$ (C$,2)
1430 C1 =  VAL (A$):C2 =  VAL (
B$):D$(C1,C2) = C$
1440  IF C1 > CX THEN CX = C1
1450  IF C2 > RX THEN RX = C2
1460  GOTO 1410
1470  PRINT D$;"CLOSE"
1480  HOME :CC = 1:CR = 1:CS =
1:RS = 1:MO = 1: GOSUB 70: GOTO
 320
```



O JOGO DA SENHA

- AS REGRAS DO JOGO
- DEFINIÇÃO DAS CORES
- O COMPUTADOR DÁ AS DICAS
- ESCOLHA A MELHOR ESTRATÉGIA

Aceite o desafio lançado por seu microcomputador neste clássico jogo de lógica. Será que você é capaz de descobrir as cores sorteadas pela máquina, na ordem correta?

O objetivo de *Senha* é descobrir as quatro cores sorteadas pelo computador, entre as seis possíveis, além da ordem correta em que elas estão dispostas. Lembre-se de que pode ocorrer repetição de cores.

A cada tentativa, você deverá digitar as iniciais das cores — por exemplo, **RLCC** para roxo, laranja, cinza e cinza. O computador, então, responderá com um código, para ajudá-lo a decifrar o segredo. Para cada cor correta em posição errada ele imprimirá um caractere branco (ou a letra **B**), e quando a ordem estiver correta, o caractere terá a cor preta (ou a letra **P**). É lógico que a ordem dos códigos não corresponde à das cores; isto tornaria o jogo muito fácil. Você tem doze chances para decifrar a senha.

As cores usadas por cada programa são as seguintes: **A**marelo, a**Z**ul, azul-**C**laro, **V**ermelho, **M**agenta e **B**ranco para o micro Spectrum; **A**marelo, a**Z**ul, **V**ermelho, azul-**C**laro, **M**agenta e **L**aranja para o TRS-Color; **A**marelo, **V**ermelho, **R**oxo, **L**aranja, **M**agenta e **C**inza para o TK-2000, o Apple e o MSX.



```
10 BORDER 0: INK 0: PAPER 4:
CLS : LET N$="671254": DIM C(
4)· DIM G(4): DIM F(4,2): LET
 C$="AZVCM"
14 PRINT AT 16,0;"CORES :"'''"
A=AMARELO Z=AZUL C=AZUL CLARO
   V=VERMELHO  M=MAGENTA  B=B
RANCO"
15 FOR N=USR "A" TO USR "A"+7
: READ A: POKE N,A: NEXT N
17 DATA 0,24,60,126,126,60,24
,0
20 FOR K=1 TO 4: LET C(K)=VAL
 N$(INT (RND*5)+1): NEXT K:
LET G=1
30 INPUT "FACA A OPCAO ";B$
35 IF LEN B$<>4 THEN  GOTO 30
90 PRINT AT G,0;"OPCAO No. ";
G;AT G,14;· FOR K=1 TO 4: LET
G(K)=(7*(B$(K)="B"))+(6*(B$(K
)="A"))+(B$(K)="Z")+(2*(B$(K)
="V"))+(5*(B$(K)="C"))+(3*(B$
(K)="M"))
92 IF G(K)=0 THEN  LET K=4·
NEXT K: GOTO 30
95 PRINT  INK G(K): BRIGHT 1;
CHR$ 144;: IF K<>4 THEN
PRINT  BRIGHT 1;" ";
97 NEXT K
100 PRINT AT G,24;
110 LET N=0: LET R$=" "· FOR K
=1 TO 4: LET F(K,1)=0: LET F(K
,2)=0: IF G(K)=C(K) THEN  LET
R$=R$+" "+CHR$ 16+CHR$ 0+CHR$
144: LET F(K,1)=1: LET F(K,2)=
1: LET N=N+1
120 NEXT K
130 FOR K=1 TO 4: IF F(K,1)=1
THEN  GOTO 170
140 FOR J=1 TO 4: IF F(J,2)=1
THEN  GOTO 160
150 IF C(J)=G(K) THEN  LET R$=
```

```
R$+" "+CHR$ 16+CHR$ 7+CHR$ 144
  LET F(J,2)=1: LET J=4
160 NEXT J
170 NEXT K
180 PRINT AT G,23;R$: INK 0:
IF N=4 THEN  PRINT AT 21,0;"VO
CE ACERTOU APOS ";G;" TENTATIV
AS": GOTO 230
190 LET G=G+1: IF G<13 THEN
GOTO 30
200 PRINT "O CODIGO CORRETO ER
A ";: FOR K=1 TO 4: PRINT  INK
C(K),CHR$ 144;" ";: NEXT K
220 PRINT
230 PRINT "JOGA NOVAMENTE ?"
240 LET A$=INKEY$: IF A$=""
THEN  GOTO 240
250 IF A$="S" THEN  RUN : STOP
```



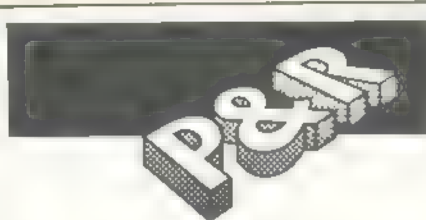
```
10 DIM C(3),G(3),F(3,1):C$="AZV
CML"
20 CLS FOR K=0 TO 3:C(K)=RND(6)
 NEXT G=1:PRINT @8,"adivinhe o
codigo"
30 PRINT @416,"TENTATIVA NO ";G
,"?  (EX: VZAC)":PRINT @448:B$=
""
40 A$=INKEY$:IF A$="" THEN 40
50 IF A$=CHR$(13) AND LEN(B$)=4
 THEN 90
60 IF A$=CHR$(8) AND LEN(B$)>0
THEN B$=LEFT$(B$,LEN(B$)-1)
70 IF LEN(B$)<4 AND INSTR(C$,A$
)<>0 THEN B$=B$+A$
80 PRINT @448,B$:GOTO 40
90 PRINT @32*G,"TENTATIVA",G;:F
OR K=0 TO 3:G(K)=INSTR(C$,MID$(
B$,K+1,1)):C=G(K)-(G(K)>3)
100 PRINT @32*G+11+K*2,CHR$(143
+C*16);:NEXT
110 N=0 R$="  ":FOR K=0 TO 3:F(
K,0)=0:F(K,1)=0:IF G(K)=C(K) TH
EN R$=R$+" "+CHR$(128):F(K,0)=1
 F(K,1)=1 N=N+1
120 NEXT
130 FOR K=0 TO 3 IF F(K,0)=1 TH
EN 170
140 FOR J=0 TO 3 IF F(J,1)=1 TH
EN 160
150 IF C(J)=G(K) THEN R$=R$+" "
+CHR$(207):F(J,1)=1:J=3
160 NEXT
170 NEXT
180 PRINT R$ IF N=4 THEN 200
190 G=G+1 IF G=13 THEN 210 ELSE
 30
200 PRINT @416," VOCE ACERTOU A
POS";G;"TENTATIVAS":GOTO 230
210 PRINT @416,"EU GANHEI. O CO
DIGO CORRETO ERA"
220 FOR K=0 TO 3 PRINT @448+K*2
," ";CHR$(143+C(K)*16-16*(C(K)>
3));:NEXT
230 PRINT:PRINT" JOGA NOVAMENTE
 ? (S/N)"
240 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 240
250 IF A$="N" THEN CLS:END ELSE
 20
```



```
10  DIM C(3),G(3),F(3,1):C$ = "
AVRLMC"
20  HOME : FOR K = 0 TO 3:C(K)
=  INT (6 *  RND (1) + 1): NEXT
 K
25 G = 1: PRINT  TAB( 6)"ADIVIN
HE A SENHA (CORES:AVRLMC)"
30  VTAB (23): PRINT "TENTATIVA
 NUMERO ";G;" (EX:AVRL):":B$ =
""
40  GET A$
50  IF A$ =  CHR$ (13) AND  LEN
 (B$) = 4 THEN 90
60  IF A$ =  CHR$ (8) AND  LEN
(B$) > 0 THEN B$ =  LEFT$ (B$,
LEN (B$) - 1)
70  IF  LEN (B$) < 4 THEN B$ =
B$ + A$
80  VTAB (G + 2): HTAB (21): PR
INT B$: GOTO 40
90  VTAB (G + 2): PRINT "CHANCE
 NUMERO ";G;
92  FOR K = 0 TO 3: FOR I = 1 T
O 6
94  IF  MID$ (B$,K + 1,1) =  MI
D$ (C$,I,1) THEN G(K) = I: GOTO
 100
96  NEXT I
98 G(K) = 0
100 C = G(K) - (G(K) > 3): NEXT
 K
105 N = 0:R$ = "  "
110  FOR K = 0 TO 3:F(K,0) = 0:
F(K,1) = 0: IF G(K) = C(K) THEN
 R$ = R$ + " " + "P":F(K,0) = 1
:F(K,1) = 1:N = N + 1
120  NEXT
130  FOR K = 0 TO 3: IF F(K,0)
= 1 THEN 170
140  FOR J = 0 TO 3: IF F(J,1)
= 1 THEN 160
150  IF C(J) = G(K) THEN R$ = R
$ + " " + "B":F(J,1) = 1:J = 3
160  NEXT J
170  NEXT K
180  HTAB (28): PRINT R$: IF N
= 4 THEN 200
190 G = G + 1: IF G = 13 THEN 2
10
195  GOTO 30
200  VTAB (22): PRINT " VOCE AC
ERTOU A SENHA EM ";G;" CHANCES"
: GOTO 230
210  VTAB (22): PRINT " EU GANH
EI, A SENHA ERA ",
220  FOR K = 0 TO 3: PRINT  MID
$ (C$,C(K),1);: NEXT
230  HTAB (1): VTAB (23): PRINT
 " QUER JOGAR NOVAMENTE?(S/N)
 "
240  GET A$
250  IF A$ = "S" THEN  GOTO 20
260  END
```



```
10 DIM C(3),G(3),F(3,1):C$="VAR
LMC"
20 CLS:KEY OFF:FOR K=0 TO 3:C(K
)=INT(6*RND(-TIME)+1):NEXT:G=1:
```



Como chegar mais rapidamente à seqüência correta das cores?

Existe apenas um segredo: obter o máximo de informações em cada jogada. Com seis tentativas, por exemplo, utilizando somente uma cor em cada, você descobrirá facilmente as quatro cores da senha, mas nada saberá sobre suas posições. Assim, convém ter sempre em vista ambos os objetivos. Talvez você perca mais tempo para chegar às cores corretas, mas a definição de suas posições na seqüência será bem mais rápida.

```
PRINT TAB(4) "ADIVINHE A SENHA
(CORES VARLMC)"
30 LOCATE 3,22  PRINT "TENTATIV
A NO";G; "? (EX:AVRL)":B$=""
40 A$ INKEY$ IF A$="" THEN 40
50 IF A$ CHR$(13) AND LEN(B$)=4
 THEN 90
60 IF A$=CHR$(8) AND LEN(B$)>0
THEN B$=LEFT$(B$,LEN(B$)-1)
70 IF LEN(B$)<4 AND INSTR(C$,A$
)<>0 THEN B$=B$+A$
8( LOCATE 24,G+4 PRINT B$ GOT(
40
90 LOCATE 3,G+4 PRINT"Tentativa
 número ";G;:FOR K=0 TO 3 G(K)
INSTR(C$,MID$(B$,K+1,1)) C=G(K)
 (G(K)>3) NEXT K
110 N=0 R$="  " FOR K=0 TO 3 F(
K,0)=0 F(K,1)=0 IF G(K)=C(K) TH
EN R$=R$+" "+"P" F(K,0)=1 F(K,1
)=1 N N+1
120 NEXT
130 FOR K=0 TO 3 IF F(K,0) 1 TH
EN 170
140 FOR J=0 TO 3:IF F(J,1)=1 TH
EN 160
150 IF C(J)=G(K) THEN R$=R$+" "
+"B":F(J,1)=1:J=3
160 NEXT
170 NEX1
180 LOCATE 28,G+4 PRINT R$ IF N
=4 THEN 200
190 G=G+1:IF G=13 THEN 210 ELSE
 30
200 LOCATE 1,21 PRINT"Você acer
tou a senha em ";G;" chances":G
OTO 230
210 LOCATE 1,21 PRINT "Eu venci
 a senha correta era ",
220 FOR I=0 TO 3 PRINT MID$(C$,
C(I),1);:NEXT
230 LOCATE 1,22 PRINT"Quer joga
r novamente? (S/N)    "
240 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 240
250 IF A$="N" THEN CLS:END:ELSE
 20
```

MAIS SOBRE PÁGINAS GRÁFICAS

- LIMITAÇÕES DA MEMÓRIA
- TÉCNICAS DE PAGINAÇÃO
- PÁGINAS GRÁFICAS E A EXIGÊNCIA DE ESPAÇO
- ANIMAÇÕES GRÁFICAS

Depois de examinar os princípios da paginação gráfica, vamos, agora, explorar suas possibilidades. Os programas deste artigo não deixarão dúvidas quanto à sua utilidade.

Como vimos no artigo da página 1 096, a paginação gráfica — a técnica de mostrar várias telas gráficas em seqüência — apresenta um grande potencial em diversos tipos de aplicação. Além de seu emprego em animação computadorizada, as páginas gráficas podem ser úteis em áreas mais "sérias", tais como planilhas financeiras ou quaisquer outros projetos que exijam a rápida mudança de uma tela cheia de informações para outra.

Já tivemos um contato inicial com essa técnica. Agora, trataremos de examinar mais de perto suas possibilidades. Antes disso, no entanto, convém recapitular alguns pontos fundamentais. A paginação gráfica consiste, basicamente, em armazenar cada tela em uma parte da memória e depois recuperá-las uma a uma. Essas telas podem conter diferentes tipos de informação: gráficos de alta ou baixa resolução, textos, ou até uma combinação de ambos. Depois de armazenadas, elas podem ser chamadas em sequência, sem que se perca tempo com o processo de montagem do desenho. O computador não precisará, portanto, executar uma série de funções demoradas, como a função **SIN**, por exemplo. Ele apenas transferirá dados de uma parte da memória para outra.

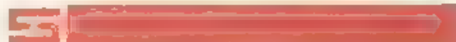
É claro que o processo de montagem do desenho continua sendo necessário em algum ponto, mas só uma vez para cada tela. Depois de prontas, elas podem ser recuperadas instantaneamente, sempre que você quiser.

LIMITAÇÕES DA MEMÓRIA

Cada página gráfica requer uma certa quantidade de memória. Essa quantidade varia de acordo com a complexidade do desenho — quanto mais cores forem usadas e quanto maior for a resolução gráfica, mais memória será exigida. Isso impõe severas limitações aos microcomputadores e, em alguns casos, temos mesmo que sacrificar um pouco a qualidade do desenho para obter mais páginas gráficas.

Tomando as medidas necessárias para economizar memória, você poderá conseguir bastante espaço para as páginas. Mas não se esqueça de que o próprio programa ocupará uma parte desse espaço. Portanto, antes de iniciar os trabalhos, calcule cuidadosamente a quantidade de memória que você ocupará com suas telas. Verifique se não é aconselhável baixar um pouco a resolução do desenho ou diminuir o número de cores. Com o planejamento, você não correrá o risco de precisar de mais espaço do que o disponível na RAM.

A técnica de paginação gráfica permite o acesso a cada tela na memória do computador. Dependendo do espaço disponível, pode-se definir previamente uma seqüência de oito ou mais telas. Para economizar espaço, lance mão de alguns pequenos truques. Por exemplo: em uma seqüência como a da página 1144, que mostra uma figura (o "homem-rabisco"), as páginas gráficas seriam exibidas na ordem 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 e assim por diante. Porém, como você deve ter observado, existem dois pares muito semelhantes: o par 2 e 4 e o par 3 e 5. Ora, em uma situação onde qualquer espacinho está a prêmio, será bem melhor armazenar apenas uma figura de cada um desses pares. Durante a projeção das imagens, nem se notará a diferença e você terá economizado um bom espaço. Usando a seqüência 1, 2, 3, 2, 3, 1, você continuará com uma animação de cinco imagens, armazenando apenas três telas.



O Spectrum de 48 K pode manipular, no máximo, oito ou nove páginas gráficas diferentes, com apenas duas cores



(INK e PAPER). Esse micro também apresenta uma limitação que envolve o tamanho da tela. Se utilizarmos dois terços dela, cada página gráfica ocupará 4 K. Somando mais 2 K, exigidos pelo programa, chegamos logo ao limite máximo de memória. Levando em conta essas limitações, o programa que se segue trabalha com oito telas, mostrando uma estrada em perspectiva.

```
10 BORDER 0: PAPER 0: INK 0 [illegible]
CLS
20 CLEAR 27999
30 GOSUB 170
40 LET [illegible]: LET [illegible]
50 FOR n=1 TO 20: PLOT RND*[illegible]
255),RND*[illegible]
60 FOR [illegible]
70 FOR [illegible]
[illegible]
      NEXT n
80 GOSUB 260
90 GOSUB 220 [illegible]
+16
100 NEXT [illegible]
110 LET [illegible]
120 FOR [illegible]
130 GOSUB [illegible]
[illegible]
140 PAUSE [illegible]
150 NEXT [illegible]
160 GOTO 110
170 DATA [illegible]
237,176,201
180 FOR [illegible]
190 READ [illegible]
200 NEXT [illegible]
210 RETURN
220 POKE [illegible]
230 POKE 28000,[illegible]
240 RAND USR [illegible]
250 RETURN
260 PLOT [illegible]
270 PLOT [illegible] DRAW [illegible]
280 PLOT [illegible] DRAW [illegible]
290 FOR [illegible]
[illegible]
300 READ [illegible]
310 [illegible]
[illegible]
320 RETURN
330 DATA [illegible]
[illegible]
[illegible]
[illegible]
[illegible]
10,1,140,105,4,-4,128,120,1,
[illegible]
[illegible]
[illegible]
[illegible]
-1,10,1
350 DATA 116,120,1,[illegible],160,100,
[illegible],4,80,100,0,[illegible],110,138,10,
10,3,130,110,[illegible],1,160,80,6,
[illegible]
```

O programa começa inicializando a tela em preto e branco e definindo a RAMTOP em 27999. A rotina das linhas 170, 180, 190, 200 e 210 coloca uma pequena rotina em linguagem de máquina acima da RAM acessível. Essa rotina, muito rápida, será a responsável pela transferência do bloco de informações da tela para a memória, à medida que as imagens vão sendo construídas. Posteriormente, irá também recuperá-las, trazendo-as de volta à tela. A linha 40 estabelece os valores dos bytes mais significativos das variáveis [illegible] e dest, que contêm, respectivamente, o endereço da [illegible] e [illegible] do seu destino.

A primeira parte [illegible] começa na linha 50 [illegible] simplesmente faz desenhos em posições aleatórias da tela. Esses desenhos são fixos e, [illegible]

[illegible]

da estrada (linha 270), a própria estrada e os postes (linha 280) e uma ave em vôo (linha 290) — sempre lendo as informações que estão nas linhas DATA no final do programa (linha 320 em diante).

Retornando da rotina de desenho, o programa é desviado para uma subrotina de POKE, na linha 220. Esta copia os 4 K de cada tela em um local apropriado da memória. O endereço dest é incrementado com o valor 16 no seu byte mais significativo (16*256=4 K), a fim de criar o espaço para a próxima página gráfica. O programa volta, mais uma vez, para a rotina de desenho e todo o processo se repete oito vezes, sendo que, a cada vez, a figura [illegible] uma posição adiante.

Em seguida, o programa [illegible] as telas sucessivamente, [illegible]

[illegible] qual a rotina da linha 220 [illegible] cada página gráfica.

T

O TRS-Color apresenta evidente vantagem [illegible] em virtude de seu poderoso comando PCOPY, que permite que o [illegible] optamos pela modalidade gráfica PMODE 3. Cinco páginas gráficas [illegible]

```
[illegible]
CIRCLE(128,120),20,[illegible]
LINE(119,116)-(127,120),PSET[illegible]
[illegible]
153),2,4
DRAW"BM137,[illegible]":PAINT(1[illegible]
[illegible]:DRAW"C3[illegible]"
DRAW"BM1[illegible]":PAINT(90,120),3,[illegible]
COLOR [illegible] LINE [illegible]
NEXT
DRAW"[illegible]":DRAW"BM141,[illegible]"+STR$(INT(86+1.5*K*K))+"D2F2DL4[illegible]2D4
DRAW"BM"+STR$(INT(141+1.8*K[illegible]
[illegible]
NEXT
FOR [illegible]
[illegible]
```

O programa começa reservando quatro blocos (1,5 K cada) para as informações da tela, pois, como selecionamos a modalidade **PMODE** 3, cada tela requisitará 6 K de memória.

Até agora utilizamos 6 K. Como o BASIC e a tela de textos ocuparão mais 1,5 K, ficaremos com 25 K de memória RAM aproveitável. Usando páginas inteiras que requerem 6 K cada uma, poderíamos contar com mais de quatro telas de páginas gráficas (25 K divididos por 6) — mas isto deixaria muito pouco espaço para o programa.

Se limitássemos o desenho a três quartos da tela, cada página iria ocupar apenas 4,5 K de memória. Utilizaríamos, assim, cinco páginas, deixando livres 2 K para o programa. Não haveria, entretanto, espaço para o sistema de operação de disco.

Depois de estabelecer **PMODE** 1 — quatro cores com uma resolução de 128 por 192 pontos —, a primeira linha do programa limpa a parte menos importante de armazenamento de strings acima do endereço 9215. Para suas próprias rotinas, você deverá defini-la somente por tentativa e erro.

A segunda linha continua com o processo de inicialização, definindo o modo de resolução da tela e a cor. Aqui se inicia também o laço de desenho, cujo primeiro comando é um **PCLS**.

A rotina de desenho que se segue ocupa grande parte do programa. A linha 30 constrói e pinta o corpo da bomba de movimento contínuo, que forma a base da figura. A linha 40 monta o funil e o preenche com cor, enquanto a linha 50 faz o mesmo com o cano. As linhas 60, 70 e 80 encarregam-se da queda da água. As linhas 90, 100 e 110 desenham as listas que ajudam a dar a impressão de que a água está em movimento. Outros detalhes da bomba — como a rotação da pá — são executados pelas linhas 120, 130 e 140.

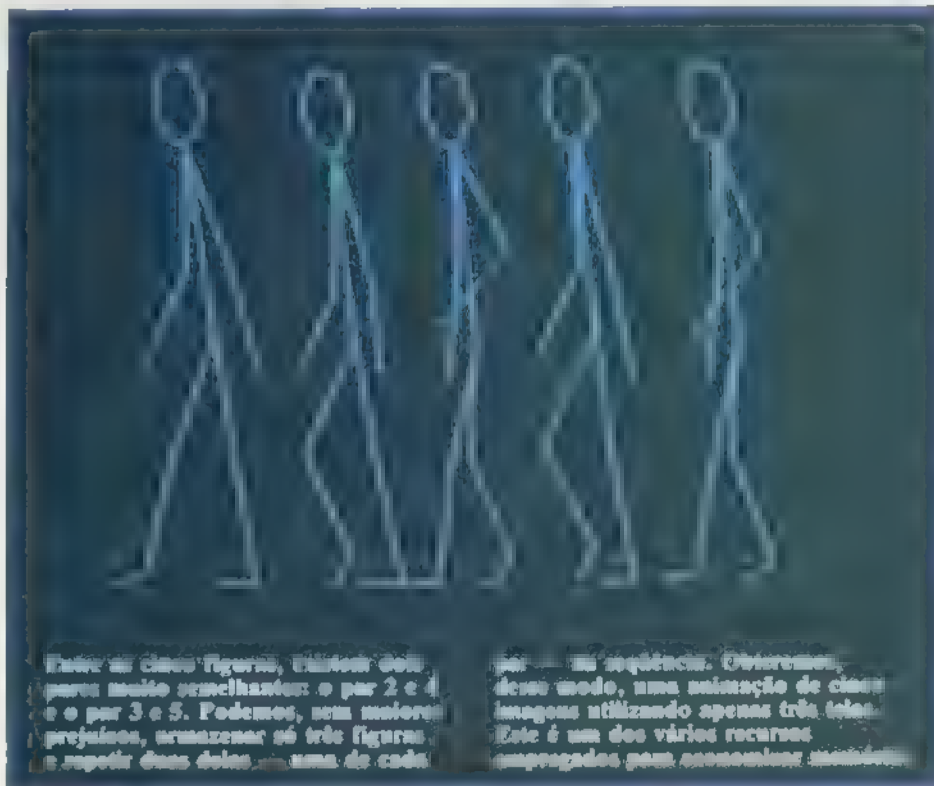
A linha 150 copia os três quartos da tela na memória — mais precisamente na área definida pela linha 10 — e o programa volta para a rotina de desenho para que sejam montadas as outras páginas gráficas. Cada uma delas é guardada na memória quando o programa alcança, novamente, a linha 150.

Depois que todas as páginas estiverem armazenadas, o programa entra na rotina de animação (linha 160). Ela mostra todas as telas, em seqüência, até o programa ser interrompido.



Quando usamos um comando gráfico do tipo **LINE**, por exemplo, o computador "rabisca" o traço pedido sobre a tabela de padrões da VRAM. Portanto, se fizermos uma cópia dessa tabela na RAM, poderemos transferi-la posteriormente para a VRAM e o desenho [illegible]

```
[illegible]
```



Entre as cinco figuras, existem dois pares muito semelhantes: o par 2 e 4 e o par 3 e 5. Podemos, sem maiores prejuízos, armazenar só três figuras e repetir duas delas [illegible] uma de cada par [illegible] na seqüência. Obteremos, desse modo, uma animação de cinco imagens utilizando apenas três telas. Este é um dos vários recursos empregados para economizar memória.

rá imediatamente refeito. Poderíamos também armazenar a tabela de cores, mas isso acrescentaria 6144 bytes aos 6144 bytes da tabela de padrões, o que nos limitaria ao uso de, no máximo, uma página gráfica. Para que nossa tela não fique sem cor, carregaremos a tabela de cor com um número específico que definirá as cores de fundo e de frente. Nosso desenho (um beija-flor) terá, portanto, apenas duas cores. Em compensação, economizamos o suficiente para três páginas gráficas.

Para transferir os dados da VRAM para a RAM, recorreremos a uma rotina da ROM chamada **LDIRMV**. Para fazer o contrário, usaremos uma outra rotina, a **LDIRVM**. Finalmente, para carregar a tabela de cores com um byte específico, utilizaremos a rotina **FILVRM**, que preenche qualquer área da VRAM com um determinado valor. O acesso a essas rotinas requer a introdução de alguns valores nos registradores da máquina. Por isso, construiremos uma pequena rotina em linguagem de máquina para chamar cada rotina da ROM.

```
10 CLS
20 CLEAR200,40960!
30 DEFUSR=40960!
40 DEFUSR1=40973!
50 DEFUSR2=40986!
60 FOR R=0 TO 37
70 READ A
80 POKE (40960!+R),A
90 NEXT
100 SCREEN2
110 X1=112:Y1=67:N=16
120 GOSUB 430
130 A=USR(0)
140 SCREEN2
150 X1=115:Y1=83:N=19
160 GOSUB 430
170 POKE 40964!,0
180 POKE 40965!,185
190 A=USR(0)
200 SCREEN 2
210 X1=158:Y1=182:N=17
220 GOSUB 430
230 POKE 40964!,0
240 POKE 40965!,209
250 A=USR (0)
260 POKE 40990!,1*16+11
270 A=USR2(0)
280 FOR I=1 TO 3:ON I GOSUB 310
,350,390:NEXT
290 FOR I=3 TO 1 STEP -1:ON I G
OSUB 310,350,390:NEXT
300 GOTO 280
310 POKE 40974!,0
320 POKE 40975!,161
330 A=USR1(0)
340 RETURN
350 POKE 40974!,0
360 POKE 40975!,185
370 A-USR1(0)
380 RETURN
390 POKE 40974!,0
400 POKE 40975!,209
410 A=USR1(0)
420 RETURN
430 FOR I=1 TO N
440 READ X2
450 READ Y2
460 LINE(X1,Y1)-(X2,Y2)
470 LET X1=X2:LET Y1=Y2
480 NEXT I
490 RETURN
500 DATA 33,00,00,17,00,161,01,
00,24,205,89,00,201
510 DATA 33,00,161,17,00,00,01,
00,24,205,92,00,201
520 DATA 33,00,32,62,00,01,00,2
4,205,86,00,201
530 DATA 86,3,79,67,90,86,192,3
,176,64,141,99,198,166,170,186
540 DATA 115,186,118,138,86,118
,12,191,48,139,51,109,67,90,90,
86
550 DATA 96,64,54,48,83,86,115,
83,160,80,240,102,185,115
560 DATA 144,113,160,144,192,14
7,179,169,144,185,115,148
570 DATA 86,118,12,191,48,139,5
1,109,67,90,83,86
580 DATA 182,160,224,121,166,12
8,105,83,90,128,160,191
590 DATA 150,116,105,83,67,90,5
1,109,48,139,10,190
600 DATA 89,118,90,128,87,137,8
3,176,70,134
```

A linha 20 reserva uma área da memória para as rotinas em linguagem de máquina. A linha 30 define o início da rotina que irá chamar a **LDIRMV** e a linha 40, o início da rotina que acessa a **LDIRVM**. A linha 50 faz o mesmo para a rotina **FILVRM**.

O laço entre as linhas 60 e 90 lê as linhas **DATA** 500, 510 e 520, que contêm as rotinas — na ordem em que foram citadas —, colocando-as na memória do microcomputador.

A rotina de desenho localiza-se entre as linhas 100 e 270. A cada página, o computador inicializa as variáveis **X1**, **Y1** e **N**, que guardam as coordenadas iniciais do desenho e o número de pontos a serem lidos. O programa é então desviado para a sub-rotina entre as linhas 430 e 490, que lê as informações contidas nas linhas 530 a 600 e traça as retas. Assim que a figura estiver completa, a rotina que armazena a tela na RAM é chamada.

O processo é o mesmo para as outras duas páginas, com uma diferença: dois comandos **POKE** irão alterar a sub-rotina de armazenamento. Esta continha, inicialmente, o endereço a partir do qual a primeira tela seria guardada na RAM. É claro que não podemos usar esse mesmo endereço para as duas outras páginas gráficas. O papel dos comandos **POKE** é, portanto, fazer a modificação necessária para o início da segunda e da terceira páginas.

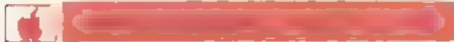
Depois que todas as telas já tiverem sido armazenadas, a linha 260 determina o valor do byte que preencherá a ta-

bela de cores, colocando-o na rotina que acessa a **FILVRM**. O valor 1 é a cor de frente — preto — e o valor 11 é a cor de fundo — amarela. A linha 270 chama essa rotina.

Em seguida chega-se à rotina encarregada de alternar as telas. Os dois laços (linhas 280 e 290) determinam uma sequência de telas do tipo 1, 1, 2, 3, 3, 2, 1, 1, fazendo com que um beija-flor em vôo permaneça mais tempo com as asas para cima e para baixo do que na posição intermediária.

Conforme o valor da variável **I**, o programa é desviado para uma sub-rotina. Nesta, um par de comandos **POKE** modifica a rotina que busca os dados na RAM. Esses comandos definem os endereços a partir dos quais a memória será transferida para a tela.

Se você quiser construir suas próprias animações, elimine a sub-rotina da linha 430, assim como todas as linhas que a chamam. Além disso, apague também a linha **DATA** 530 e as que a seguem e coloque seus comandos gráficos a partir da linha que inicializava as variáveis, tendo o máximo cuidado para não provocar sobreposições.



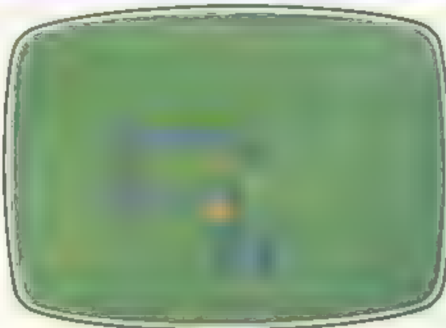
Como você deve saber, os micros da linha Apple oferecem ao usuário duas páginas gráficas prontas para serem utilizadas. Por meio dos comandos usuais do BASIC, não podemos escrever em uma das páginas sem que ela apareça no vídeo. Se isto fosse possível, eliminaríamos, em uma animação, aquele efeito desagradável provocado pelos comandos **HGR** e **HGR2** para limpar a tela — enquanto mostrássemos uma tela, estaríamos simultaneamente limpando e escrevendo na outra. Em seguida, bastaria trocar as duas e repetir o processo até o fim da animação. Teríamos, assim, uma velocidade moderada, mas não haveria um limite de páginas gráficas.

O próximo programa ilustra o desenvolvimento desse processo com a animação de uma figura — o homem-rabisco —, comentada anteriormente.

```
10  POKE  - 16304.0:X = 200
20  HGR · HGR2 · HCOLOR= 3
30  POKE 230,32  GOSUB 240
40  POKE  - 16300,0
50  POKE 230,64· GOSUB 280
60  POKE  - 16299,0
70  POKE 230,32  HCOLOR= 0· GOS
UB 240· HCOLOR= 3· GOSUB 320
80  POKE  - 16300,0
90 P1 = 32:P2 = 64:M1 =  - 1630
0:M2 =  - 16299
100  POKE 230,P2· HCOLOR= 0· GO
SUB 280.X = X - 15: HCOLOR= 3:
GOSUB 280
110  IF X < 30 THEN  STOP
120  POKE M2,0
130  POKE 230,P1:X = X + 15: HC
OLOR= 0· GOSUB 320 X = X - 15
HCOLOR= 3· GOSUB 320
140  POKE M1,0
150  POKE 230,P2  HCOLOR= 0  GO
SUB 280· HCOLOR= 3· GOSUB 240
160  POKE M2,0
170  POKE 230,P1· HCOLOR= 0· GO
SUB 120  HCOLOR= 3  GOSUB 280
180  POKE M1,0
190  POKE 230,P2  HCOLOR= 0· GO
SUB 240: HCOLOR= 3: GOSUB 320
200  POKE M2,0
210 TR = P1:P1 = P2:P2 = TR
220 TR = M1·M1 = M2:M2 = TR
230  GOTO 100
240  HPLOT X - 16,136 TO X - 9,
133 TO X,94 TO X + 8,136 TO X,1
36
250  HPLOT X,94 TO X - 3,58 TO
X - 9,50 TO X,48 TO X - 3,58
260  HPLOT X - 10,97 TO X - 4,8
2 TO X - 3,58 TO X + 9,98
270  RETURN
280  HPLOT X - 15,136 TO X - 11
,132 TO X - 10,114 TO X - 1,97
TO X + 7,136 TO X,136
290  HPLOT X - 1,97 TO X - 7,61
 TO X - 13,50 TO X - 4,48 TO X
- 7,61
300  HPLOT X - 12,97 TO X - 6,8
5 TO X - 7,61 TO X + 1,82 TO X
+ 2,97
310  RETURN
320  HPLOT X - 15,136 TO X - 8,
136 TO X - 5,114 TO X - 1,94 TO
 X - 5,114 TO X + 3,132 TO X -
3,136
330  HPLOT X - 1,94 TO X - 5,61
 TO X   12,50 TO X   3,48 TO X
- 5,61
340  HPLOT X - 7,98 TO X - 3,61
 TO X + 5,78 TO X   8,90
350  RETURN
```



A estrada sem fim no Spectrum.



TRS-Color: bomba moto-contínua.

ANIMAÇÃO GRÁFICA

As instruções para as telas de números 1, 2 e 3 se encontram, respectivamente, a partir das linhas 240, 280 e 320. Elas serão chamadas na ordem 1, 2, 3, 2, 3, 1. Se quisermos mostrar um desenho na página 2, por exemplo, teremos que executar as etapas do processo descrito em seguida, enquanto a página 1 está na tela:

- habilitar a escrita na página 2
- apagar a página 2
- escrever na página 2
- mostrar a página 2 na tela

Para habilitar a escrita em uma página, é necessário colocar um determinado valor no endereço 230 por meio de um comando **POKE**. Se esse valor for 32, qualquer comando gráfico será executado na página 1; se for 64, será acessada a página 2.

Para apagar uma página gráfica, temos simplesmente de descolorir a figura, utilizando a mesma rotina que a montou, com **HCOLOR = 0**.

Empregamos também o comando **POKE** para colocar uma página na tela. Se esse comando for dado no endereço −16300, será mostrada a página 1; se f[illegible]o no endereço −16299, será mostrada a página 2. Em ambos os casos, porém, deve haver um comando **POKE** no endereço −16304; caso contrário, será mostrada a página de texto.

A fase de inicialização localiza-se entre as linhas 10 e 90, onde ocorrem as primeiras impressões. A variável **X** controla a posição horizontal da figura na tela e será incrementada com o valor 15 a cada passo.

Entre as linhas 100 e 230 situa-se o laço que irá controlar o movimento do homem-rabisco até o fim da animação. Na linha 100 e na linha 130, subtrai-se 15 dessa variável para que seja apagada a imagem que foi feita antes do incremento.

As linhas 210 e 220 promovem a troca de valores entre as variáveis **P1**, **P2**, **M1** e **M2**. Estas definem as páginas em que serão feitos os desenhos e as páginas que serão mostradas. A troca é necessária para que não se perca a alternância de telas.

AVALANCHE: A ESCALADA

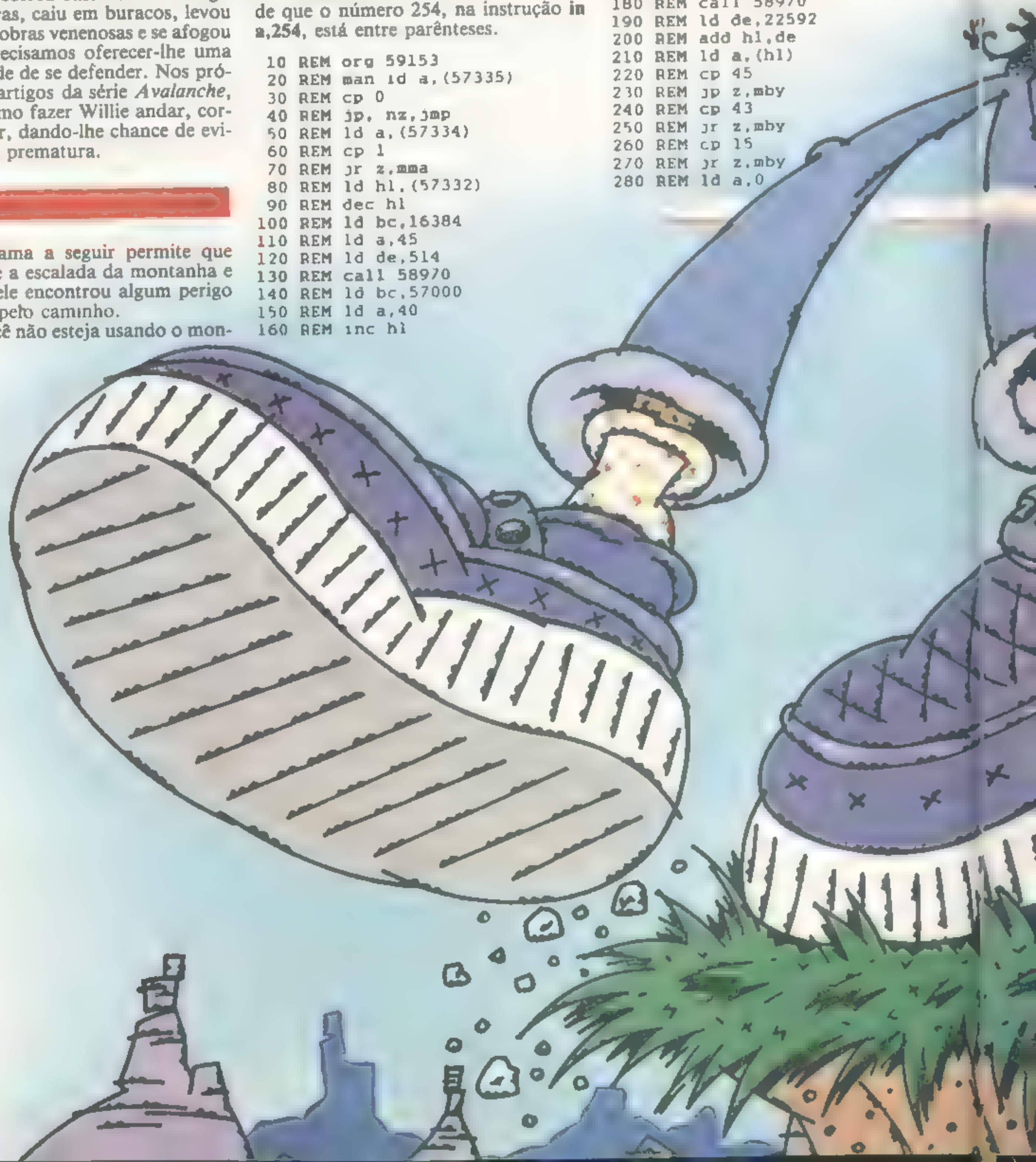
Até agora, nosso personagem permaneceu indefeso diante dos perigos que o cercam. Vamos dar-lhe a chance de escapar, fazendo com que escale a montanha, corra e salte.

Willie já sofreu bastante: foi atingido por pedras, caiu em buracos, levou picadas de cobras venenosas e se afogou no mar. Precisamos oferecer-lhe uma oportunidade de se defender. Nos próximos três artigos da série *Avalanche*, veremos como fazer Willie andar, correr ou saltar, dando-lhe chance de evitar a morte prematura.

O programa a seguir permite que Willie inicie a escalada da montanha e verifica se ele encontrou algum perigo ou prêmio pelo caminho.

Caso você não esteja usando o montador Assembler de INPUT, lembre-se de que o número 254, na instrução **in a,254**, está entre parênteses.

```
 10 REM org 59153
 20 REM man ld a,(57335)
 30 REM cp 0
 40 REM jp, nz,jmp
 50 REM ld a,(57334)
 60 REM cp 1
 70 REM jr z,mma
 80 REM ld hl,(57332)
 90 REM dec hl
100 REM ld bc,16384
110 REM ld a,45
120 REM ld de,514
130 REM call 58970
140 REM ld bc,57000
150 REM ld a,40
160 REM inc hl
170 REM ld de,258
180 REM call 58970
190 REM ld de,22592
200 REM add hl,de
210 REM ld a,(hl)
220 REM cp 45
230 REM jp z,mby
240 REM cp 43
250 REM jr z,mby
260 REM cp 15
270 REM jr z,mby
280 REM ld a,0
```



```
290 REM in a,254
300 REM bit 2,a
310 REM jr nz,mft
320 REM ld b,1
330 REM bit 3,a
340 REM jr nc,mlj
350 REM ld b,129
360 REM mlj ld a,b
370 REM ld (57335),a
380 REM jr mct
390 REM mft bit 3,a
400 REM jr nz,mct
410 REM ld a,1
420 REM ld(57334),a
430 REM mct ld hl,(57332)
440 REM ld de,191
450 REM sbc hl,de
460 REM jr nc,mor
470 REM ld a,1
480 REM ld (57336),a
490 REM mor ret
500 REM mma ld de,3
510 REM ld hl,1548
520 REM call 949
530 REM ld hl,(57332)
540 REM ld de,22561
550 REM add hl,de
560 REM ld a,(hl)
570 REM cp 43
580 REM jr z,mby
590 REM cp 44
600 REM jr z,mts
610 REM cp 42
620 REM jr z,mby
630 REM ld de,32
640 REM add hl,de
650 REM ld a,(hl)
660 REM cp 15
670 REM jr z,mby
680 REM cp 45
690 REM jr z,mby
700 REM cp 43
710 REM jr z,mby
720 REM ld hl,(57332)
730 REM ld a,40
740 REM ld bc,57016
750 REM ld de,514
760 REM call 58970
770 REM inc hl
780 REM ld (57332),hl
790 REM mts ld a,0
800 REM ld (57334),a
810 REM ret
820 REM mby ld a,2
830 REM ld (57336),a
840 REM ret
850 REM jmp ret
```

Você pode testar essas linhas mesmo que ainda não tenha na memória as outras duas rotinas de movimentação de Willie. Uma instrução **ret**, que será apagada mais tarde, foi colocada no fim da rotina que acabamos de listar. Em conseqüência, se você chamar uma rotina inexistente, o programa apenas retorna e nenhum erro ocorre.

QUAL É O MOVIMENTO?

A variável na posição de memória 57 335 informa se Willie irá ou não pular. O conteúdo desta posição é carregado no acumulador e comparado com 0. Se não for igual a 0, Willie irá pular. O processador salta, então, para a rotina **jmp**, que ainda não foi publicada. Como, em seu lugar, encontra apenas uma instrução **ret**, volta para o local onde a rotina foi chamada.

Se Willie não vai pular (o conteúdo de 57335 é 0), o processador continua a execução da rotina. O conteúdo da posição de memória 57334 é carregado no acumulador. Essa posição contém a chamada *variável da caminhada*.

Existem duas figuras para Willie — numa delas, ele está com as pernas juntas e, na outra, com as pernas abertas. Enquanto nosso personagem não se locomove, a primeira figura é impressa continuamente no mesmo lugar. Mas, quando ele está andando, as duas figuras são impressas alternadamente. A variável da caminhada informa ao processador se Willie deve andar uma posição — nesse caso, a figura adequada é a que tem as pernas abertas.

A instrução **cp 1** verifica qual figura será impressa. Se for a de valor 1 — que tem as pernas abertas —, a instrução **jr z,mma** manda o processador para a rotina que faz Willie andar e imprime essa figura. Se for a de valor 0 — que tem as pernas fechadas —, o processador continua.

O HOMEM INVISÍVEL

Ao andar, Willie abre as pernas e se desloca uma posição à frente, onde aparece com as pernas juntas. Portanto, o algoritmo que produz esse efeito imprime, em seqüência, as figuras 0, 1 e 0.

Como a figura que tem as pernas abertas ocupa duas posições adjacentes e a que tem as pernas fechadas, só uma, obtém-se um movimento bem suave.

Para imprimir a figura 0, é preciso apagar a figura anterior — do contrário, teremos várias partes de Willie pela tela. A instrução **ld hl,(57332)** carrega a posição de Willie — armazenada nos endereços 57332 e 57333 — no par HL. Esse valor é decrementado para voltar uma posição.

O par BC é então carregado com

16384. Este é o endereço do topo da tela, onde se encontra o padrão de céu. A é carregado com 45 e DE, com 514. A rotina de impressão de blocos em 58970 é chamada em seguida. Como você deve se lembrar, o par HL, nessa rotina, contém a posição da tela, e o par BC, o apontador de dados. A, por sua vez, especifica a cor — 45 é ciano sobre ciano — e o par DE fixa o tamanho do bloco. O número 514 fornece um bloco de dois por dois caracteres. D contém o número de linhas e E, o número de colunas — 2 × 256 + 2 = 514.

Assim, quando é chamada, essa rotina imprime um bloco dois por dois de céu na posição imediatamente anterior àquela onde você irá imprimir Willie de novo. Em outras palavras, ela apaga o velho Willie que, com suas pernas abertas, ocupa quatro caracteres.

IMOBILIDADE

Agora precisamos imprimir o novo Willie com as pernas juntas. Os dados para isso começam em 57000. Logo, o par BC é carregado com 57000. O acumulador A é carregado com 40, código de azul sobre fundo ciano, a cor de Willie.

O par HL, anteriormente decrementado, volta a ser incrementado para apontar a posição do personagem. O par DE é carregado com 258 — dando um bloco de um por dois (1 × 256 + 2 = 258), o espaço que Willie ocupa com as pernas juntas. A rotina de impressão de blocos é chamada para imprimir Willie.

ONDE WILLIE ESTÁ PISANDO?

Como nosso personagem se moveu uma posição à frente, convém verificar onde ele está pisando. Se for numa cobra, num buraco ou na água, o processador deve ser informado de sua morte.

Em primeiro lugar, verifique a cor do caractere que está sob os pés de Willie. O arquivo de cores começa em 22528. O valor contido em HL corresponde à posição da tela ocupada pela cabeça de Willie. Mas, como você quer a cor do caractere que está sob seus pés, precisará adicionar o valor 64.

Por isso, 22592 é carregado no par de registros DE e somado ao par HL. O resultado da instrução **add hl,de** é sempre colocado em HL. O conteúdo da posição da memória que HL aponta no momento é colocado em A. O acumulador passa a conter a cor do caractere que está sob os pés de Willie.

Esse valor é comparado com 45 — ciano sobre ciano, a cor do céu que preenche os buracos. Se for 45, a instrução **jr z,mdy** manda o processador para uma rotina curta que faz Willie morrer. Se não há um buraco debaixo de Willie, o conteúdo do acumulador é comparado com 43 — magenta sobre ciano, a cor da cobra — e com 15 — branco sobre azul, a cor do mar. Se qualquer um desses valores estiver presente, o processador salta para a rotina **mdy** e elimina Willie. Caso contrário, o processador continua a execução da rotina — Willie está salvo.

HORA DE PULAR

Nesta parte da rotina, a ação se encerra, efetivamente, com a impressão de Willie com as pernas juntas, uma posição à frente. Se ele não morreu, o passo seguinte consiste em verificar quando ele irá se mover de novo. Para isso, empregamos o comando **in**.

Antes, porém, A é carregado com 0 — o que significa que o teclado inteiro será analisado e que qualquer conjunto de teclas poderá ser utilizado para controlar os movimentos de Willie. Embora na página de instruções M e N tenham sido especificadas, outras combinações — tais como J e K ou U e I — são apropriadas para fazer nosso personagem pular e andar.

O comando **in** é usado para efetuar uma busca na porta 254. A instrução **bit 2,a** analisa o bit dois do número encontrado para verificar se a tecla M — ou J ou U — foi pressionada. Todos os bits equivalentes às teclas têm normalmente o valor 1, mas assumem o valor 0 quando a tecla é pressionada. Assim, se M foi pressionada, o bit dois tem valor 0 e o processador ignora a instrução **jr nz**. Caso contrário, ele salta para a rotina **mft**

Se a tecla foi pressionada e Willie deve pular, B é carregado com 1. Em seguida, o bit três é testado do mesmo modo. Se N não foi pressionada e o bit três tem valor 1, a instrução **jr nz** faz o processador saltar a próxima instrução. Mas, se a tecla foi pressionada, a instrução **jr nz** não tem efeito e B é carregado com 129.

O significado dos números 1 e 129, que utilizamos para carregar B, será explicado quando apresentarmos as seções do programa que executam o salto vertical e o salto à frente.

O processador transfere o conteúdo de B para A e o carrega em 57335. Este, como você deve se lembrar, corresponde à posição de memória que o processador irá examinar no início da rotina para ver se Willie pulará ou não.

A instrução **jr mct** faz o processador saltar a próxima rotina.

WILLIE VAI ANDAR?

Se não houve pressão sobre M, o processador vai para a rotina **mft**. Ela verifica se apenas a tecla N foi pressionada — ou seja, se Willie, em vez de pular, simplesmente irá andar.

Para ver se a tecla N foi pressionada, o bit três é testado de novo. Note que o valor da porta 254 ainda está no acumulador — o processador pulou para este ponto da rotina depois que o bit dois foi testado.

Se N não foi pressionada, **jr nz** salta a rotina seguinte. Caso contrário, A é carregado com 1 e esse valor é armazenado no endereço 57334, onde se encontra a variável da caminhada (que examinamos no início da rotina para saber qual das figuras de Willie seria impressa na tela). Um valor 1 nessa posição indica que Willie deve ser impresso com as pernas abertas — em outras palavras, ele está andando.

PRÊMIOS

Há apenas mais uma verificação a fazer: nosso personagem alcançou algum prêmio?

A posição de Willie é armazenada nos endereços 57332 e 57333. O conteúdo desses endereços é carregado no par HL e o valor 191 — correspondente a posição de tela onde o prêmio foi impresso — é colocado em DE.

O conteúdo desse par de registros é subtraído do conteúdo de HL. Se até aqui Willie não tiver obtido uma recompensa, a instrução **jr nc** segue para a instrução **ret** e retorna. Caso contrário, o valor 1 é carregado no acumulador e armazenado em 57336. Esta é a posição de memória que a rotina principal verifica para saber se o escore deve ser incrementado e se é preciso começar nova tela. Feito isso, o processador encontra **ret** e retorna.

UM PASSO À FRENTE

O par DE é carregado com o valor 3 e HL, com 1548. Depois, a rotina BEEP, no endereço 949, é chamada executando o efeito sonoro da caminhada.

Antes de seguir adiante, porém, Willie deve ver o que há na sua frente. Sua posição é colocada no par HL. Adicionando 22561 a esse valor, obtemos a cor do caractere que se encontra 33 pontos

adiante do conteúdo de 57332 e 57333. Como este aponta para a cabeça de Willie, 33 é um caractere abaixo e à direita, indicando a posição imediatamente à frente de seus pés. O valor dessa posição é carregado no acumulador pela instrução **ld a,(hl)**.

A cor do caractere à frente dos pés de Willie é comparada com a cor do mar — 15, branco sobre azul — da cobra — 43, magenta sobre ciano — e da pedra — 42, vermelho sobre ciano. Se a cor do mar, da cobra ou da pedra estiverem presentes, a instrução **jr z,mdy** faz o processador saltar para a rotina que elimina Willie. Mas, se adiante dos pés de Willie estiver o caractere de encosta, o processador salta para a rotina que deixa o personagem parado — é evidente que ele não pode ir em frente a não ser pulando.

Investigar o caractere que está adiante de Willie não é tudo. Devemos verificar também em qual caractere ele estará pisando depois de se mover. Para isso, carrega-se DE com 32, que é adicionado ao conteúdo de HL, o que move o apontador 32 caracteres à frente — ou seja, para a linha de baixo.

A cor nessa posição é carregada no acumulador pela instrução **ld a,(hl)** e o conteúdo do acumulador é comparado com a cor do mar, a de um espaço vazio e a de uma cobra. Se um desses valores estiver presente, a instrução **jr z,mdy** vai para a rotina da morte.

Se nada disso aconteceu e Willie ainda está vivo, HL é carregado com a posição original do personagem. A é carregado com 40 — a cor de Willie —, BC, com 57016 — o endereço inicial dos padrões para a figura com as pernas abertas — e o par DE, com 514 — Willie com as pernas abertas ocupando um bloco de dois por dois caracteres. Em seguida, a rotina de impressão de bloco em 58970 é chamada e imprime na tela Willie com as pernas abertas.

O par HL é incrementado e carregado de volta em 57332 e 57333. Atualizamos, assim, a posição de Willie que, na próxima vez, estará um caractere à direita.

AINDA PARADO

A rotina **mts** é diretamente chamada quando Willie encontra a encosta à sua frente e não pode prosseguir a não ser pulando. Mas o processador também aciona essa rotina quando ele foi impresso com as pernas abertas.

Um valor 0 é carregado no acumulador e no endereço 57334, informando ao processador que, na próxima vez, o personagem deve ser impresso com as pernas juntas — mesmo que seja no mesmo lugar, se ele não tiver se movido, ou uma posição à frente, se foi impresso com as pernas abertas e sua posição foi incrementada.

A seguir, o processador retorna.

O ÚLTIMO SUSPIRO

Se Willie se afogou no mar, foi picado por uma cobra, caiu num buraco ou foi atingido por uma pedra, a rotina **mdy** é chamada. Para a divulgação de sua morte, coloca-se 2 no endereço 57336. Em algum ponto do programa, esse endereço será checado e o funeral de Willie será providenciado.

Depois disso, o processador retorna da rotina. O último **ret** da listagem, precedido do rótulo **jmp**, será apagado pela próxima parte de *Avalanche*. Sua única função consiste em impedir que haja erro na falta desta.

O programa a seguir faz Willie andar e verifica se ele se defrontou com algum perigo — ou recompensa.

```
10  ORG 19902
20  MAN LDD 18249
30  ANDB #31
40  CMPB #30
50  BNE MANI
60  LDA #1
70  STA 18252
80  MANI LDA 18261
90  LBNE JUM
100  LDX 18249
110  LEAX 544,X
120  LDX,X
130  CMPX #$5555
140  LBEQ MDY
150  CMPX #$AAAA
160  LBEQ MDY
170  CMPX #$5FF5
180  LBEQ MDY
190  CLRB
200  CLR 18264
210  LDA #$BF
220  STA $FF02
230  LDA $FF00
240  STA 18262
250  LDA #$DF
260  STA $FF02
270  LDA $FF00
280  STA 18263
290  CMPA #$F7
300  BNEE MANA
310  LDB #1
320  LDA 18262
330  CMPA #$F7
340  BNE MAND
350  LDB #129
360  MAND STB 18261
370  LDA 18264
380  BNE MANC
390  LDX 18249
400  PSHS X
410  BRA MMI
420  MANA LDA 18262
430  CMPA #$F7
440  BNE MAND
450  LDA #1
460  STA 18264
470  BRA MAND
480  MANC LDX 18249
490  LDU #1536
500  JSR CHARPR
510  LEAX 254,X
520  JSR CHARPR
530  LDX 18249
540  LEAX 1,X
550  PSHS X
560  LEAX 353,X
570  LDA ,X
580  CMPA #$D5
590  BEQ MDYA
600  CMPA #$FF
610  BEQ MDYA
620  CMPA #$50
630  BEQ MTS
640  LDA 18251
650  BEQ MMO
660  MMI LDX,S
670  LDU #17774
680  JSR CHARPR
690  LDX ,S
700  LEAX 256,X
710  JSR CHARPR
720  CLR 18251
730  BRA MANE
740  MMO LDX ,S
750  LDU #17814
760  JSR CHARPR
770  LDX ,S
780  LEAX 256,X
790  LDU #17846
800  JSR CHARPR
810  LDA #1
820  STA 18251
830  MANE LDX ,S
840  STX 18249
850  PULS X
860  RTS
870  MDYA PULS X
880  MDY LDA #2
890  STA 18252
900  RTS
910  MTS PULS X
920  LEAX -1,X
930  PSHS X
940  BRA MMI
950  JUM RTS
960  CHARPR EQU 19402
```

Essa rotina chama outras que ainda não estão na memória. Para evitar erros, não tente executá-la antes de ter colocado RTS nos lugares adequados.

RECOMPENSAS

A primeira parte da rotina procura saber se Willie alcançou suas recompensas. Para isso, a posição do personagem na tela é carregada no registrador D. Como existem 32 posições — 32 é 2↑5

—, precisamos investigar apenas os cinco últimos bits da posição de tela para trabalhar com a coordenada X de Willie. Se esta coordenada X tiver chegado a 30, a coordenada X do prêmio (cuja posição Y está fixa junto à encosta), Willie deve agarrá-lo.

A operação AND é feita, então, entre o conteúdo de B e 31 — B é o registrador menos significativo do par de registradores AB que compõe D. Isso isola os cinco bits menos significativos, que são comparados ao número 30.

Se os cinco últimos bits contêm 30, o valor 1 é carregado no acumulador e armazenado na variável morte, em 18252. Assim, o processador é informado de que deve trazer a próxima tela. Caso contrário, a instrução **BNE** faz o processador saltar essas instruções.

PULO OU PERIGO

A é carregado com o conteúdo de 18261, a variável do salto. Se não for 0, Willie estará saltando e o processador vai para a rotina **JUM**. Isso é feito com um desvio longo porque o rótulo está muito distante.

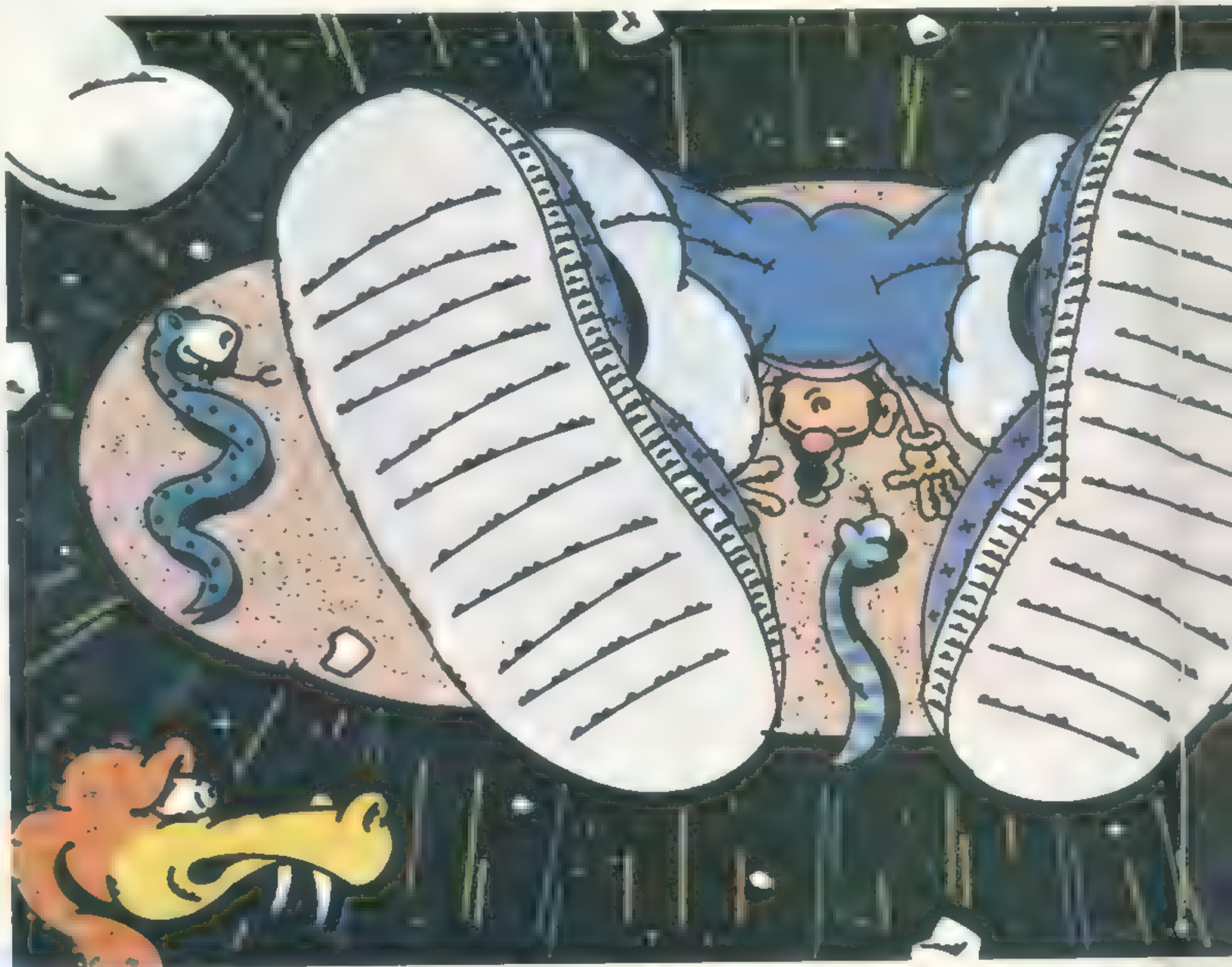
A tarefa seguinte é verificar se o personagem está diante de algum perigo fatal — um buraco, uma cobra ou o mar. A posição de Willie vai, então, de 18249 para o registrador X.

A instrução **LEAX 544,X** adiciona 544 à posição de Willie. Como ele tem dois caracteres de altura, precisamos acrescentar 2 × 256 (512) para apontar seus pés. Adicionamos 32 ao resultado, obtendo 544, porque há uma linha vazia de pontos entre os pés de Willie e o chão. O registrador X é carregado com o conteúdo dos dois bytes que estão sob os pés de Willie. Esse valor é comparado com $5555, $AAAA e $5FF5.

$5555 é o código de amarelo, que representa o céu — o que indicaria haver um buraco sob os pés de Willie; $AAAA é azul, a cor do mar, e $5FF5, vermelho sobre amarelo, a cor da cobra. Se uma dessas cores for encontrada sob seus pés, Willie está morto e a instrução **LBEQ** faz o processador saltar para a rotina **MDY**, que o elimina.

AS TECLAS

A tarefa seguinte consiste em verificar se alguma tecla foi pressionada. As



teclas M e N controlam, respectivamente, a corrida e o salto de Willie. Portanto, qualquer pressão sobre elas precisa ser detectada.

Primeiro limpa-se o registrador B, que será usado para carregar a variável que controla o salto de Willie. A posição de memória 18264 é ajustada com 0, já que irá armazenar a variável que controla a caminhada de Willie.

Para saber se uma tecla foi pressionada, deve-se analisar a matriz do teclado. No nosso caso, estamos interessados particularmente nas teclas M e N. A letra M fica na quarta coluna da sexta linha e a letra N, na quarta coluna da sétima linha.

Para examinar a situação de uma tecla, precisamos escrever o número da linha em $FF02; sua situação atual será dada em $FF00. Assim, para verificar a tecla N, armazena-se $BF em $FF02. $BF é 10111111 em binário, e é armazenado porque seleciona a sétima linha. O resultado em $FF00 é carregado no acumulador e armazenado em 18262.

M é examinada da mesma maneira, mas, desta vez, $DF — 11011111 em binário — é escrito em $FF02 e o resultado é armazenado em 18263. A instrução **CMPA #$F7** compara o resultado com $F7 ou 11110111. Lembre-se de que você está tentando ver se uma tecla da quarta linha foi pressionada — esta linha estará em zero. Se $F7 não está presente — e Willie não está saltando —, a instrução **BNE** desvia o processador para **MANA**. Mas, se $F7 estiver presente, B é carregado com 1 para indicar um salto vertical.

O resultado do exame da linha da tecla N, armazenado em 18262, é então carregado no acumulador e comparado com $F7. Observe que a tecla N está na mesma coluna que a tecla M, embora em linha diferente. Se o valor não for encontrado, **BNE** salta a instrução seguinte. Caso contrário, B é carregado com 129 para indicar que Willie dará um salto à frente. Quer seja 1, quer seja 129, o valor vai para 18261.



WILLIE ESTÁ ANDANDO?

O conteúdo da posição de memória 18264 é carregado no acumulador. Esta é a posição que armazena a variável que indica se Willie está andando ou não. Ela foi limpa, ou seja, ajustada com 0, no início do programa. Mas, neste ponto, não está necessariamente vazia porque a rotina que verifica se Willie está andando (que inicia no rótulo **MANA**) salta de volta para **MAND**.

Se o conteúdo da posição de memória 18264 não for 0 e Willie estiver andando, a instrução **BNE** faz o processador saltar para **MANC**, onde o programa começa a fazer Willie se mover. Caso contrário, o registrador X é carregado com o conteúdo de 18249 (endereço que armazena a posição de Willie na tela) e guardado na pilha. A instrução **BRA MNI** faz o processador saltar para **MNI**, que imprime o personagem na tela na posição definida pelo último valor armazenado na pilha.

Se M não foi pressionada e Willie não está pulando, o processador salta para **MANA**, para ver se ele está andando. O resultado do exame da tecla N, armazenado em 18262, é carregado de volta no acumulador e comparado com $F7. Se o valor não está presente, a instrução **BNE** manda o processador de volta para **MAND**. Como a posição de memória 18264 ainda está vazia, Willie é impresso no mesmo lugar. Isso significa que ele está efetivamente parado.

Mas, se N foi pressionada e Willie deve andar, 1 é carregado no acumulador e armazenado em 18264. Quando o processador saltar de volta para **MAND**, o personagem começará a andar.

APAGANDO A FIGURA ANTERIOR

Quando Willie começa a andar, é preciso apagá-lo de sua velha posição na tela. Do contrário, partes da figura serão deixadas pelo caminho.

X é carregado com a posição do personagem em 18249. O apontador de dados, U, é carregado com 1536, que corresponde ao canto superior esquerdo da tela. Assim, quando o processador saltar para a sub-rotina **CHARPR**, imprimirá dois caracteres de céu sobre Willie, apagando a metade superior da figura.

Para apagar a metade inferior de Willie, adiciona-se 254 a X e a rotina **CHARPR** é chamada outra vez. Soma-se 1 à posição da figura, que é carregada no registrador X — o que faz Willie se mover um caractere à frente. O resultado é armazenado na pilha.

Depois que Willie anda, precisamos verificar onde ele está pisando. A instrução **LEAX 353,X** incrementa o apontador que passa a indicar um byte imediatamente à frente das pernas do personagem, e **LDA ,X** carrega esse byte no acumulador. Ele é então comparado com $D5, a cor gráfica para a língua da cobra, e com $FF, a cor da pedra. Se algum desses valores estiver presente, Willie está condenado. A instrução **BEQ** manda então o processador para a rotina **MDYA**, que o elimina.

Esse byte também é comparado com $50, a cor gráfica correspondente à encosta da montanha. Se seu valor estiver presente, o processador salta para **MTS**, que volta a imprimir Willie no mesmo lugar, evitando que ele ande.

O QUE IMPRIMIR

A posição de memória 18251 é usada como uma baliza para informar ao processador qual das duas figuras será impressa na próxima vez — a que tem as pernas abertas ou a que tem as pernas juntas. Quando elas são impressas alternadamente, dão a impressão de que Willie está andando.

A baliza em 18251 é carregada no acumulador. Se está ajustada com 0, a instrução **BEQ** faz o processador pas-

sar para a rotina que começa com o rótulo **MNO**. Esta imprime a figura com as pernas abertas. Mas, se o conteúdo de 18251 não for 0 e a baliza estiver ajustada com 1, o salto não ocorre e o processador continua na rotina **MNI**, que imprime uma figura com as pernas juntas. Repare que, se Willie não estiver andando, o processador também pula para **MNI** e o imprime com as pernas juntas.

X é carregado com o último valor armazenado na pilha da máquina. Revendo a listagem, você constatará que o último dado colocado na pilha de máquina foi a nova posição de Willie, um caractere adiante de sua última posição. O apontador da pilha do usuário, U, que funciona como apontador de dados na rotina **CHARPR**, é carregado com 17774. Este é o endereço inicial dos dados da figura com as pernas juntas. O processador vai para a rotina **CHARPR** e imprime a metade superior de Willie.

O registrador X é carregado de novo e incrementado com 256, o que faz o apontador se mover um caractere para baixo — a posição da metade inferior de Willie. Observe que não foi preciso carregar U de novo. Ele é o apontador da pilha do usuário, e se ajusta automaticamente quando os dados são impressos na tela.

A posição de memória 18251 é ajustada com 0, para que a outra figura de Willie — a de pernas juntas — seja impressa na próxima vez. Em seguida, o processador vai para **MANE**.

Logo abaixo encontra-se a rotina **MMO**, que imprime a figura seguinte de Willie. Ela trabalha exatamente do mesmo modo. Desta vez, porém, U é ajustado com 17814, já que corresponde ao início dos dados para a figura com as pernas abertas. No final da rotina, 1 é carregado em A e armazenado em 18251, para que Willie seja impresso com as pernas juntas na próxima vez que a rotina de movimentação for chamada.

Qualquer que seja a figura impressa, o processador aciona a rotina **MANE**. Ela carrega a nova posição de nosso personagem em X e armazena-a em 18249. O último valor da pilha é colocado em X, para evitar que a pilha de máquina aumente descontroladamente. O processador então retorna.

MORTE OU IMOBILIDADE

Se Willie tiver sucumbido a algum dos perigos que o ameaçavam e estiver morto, o processador é mandado para **MDYA** ou **MDY** — o rótulo dependerá do local onde ele morreu. Se encontrou um perigo no início do programa, antes que a nova posição tivesse sido colocada na pilha, o processador irá para **MDY**. Mas, se a nova posição de Willie já foi colocada na pilha antes do passo fatal, ele irá para **MDYA**, e a nova posição será recuperada da pilha.

A é carregado com 2 — valor que indica a morte de Willie. Em seguida, esse valor é armazenado em 18252, posição da variável da morte.

Feito isso, o processador retorna.

Se Willie tiver esbarrado na encosta, o processador é mandado para **MTS**. Aqui, a nova posição de Willie é novamente recuperada da pilha, mas seu valor é utilizado e, depois de decrementado em 1, volta à pilha.

O processador salta mais uma vez para **MNI** e imprime Willie com as pernas juntas, na mesma posição de antes, o que o deixa efetivamente parado.



O programa a seguir permite que Willie inicie sua corrida para o topo da montanha e verifica se ele encontrou algum perigo ou recompensa. Caso você esteja usando o Assembler de INPUT, precisará fazer uma alteração: na linha 5010, o comando **DIM T$(100)** muda para **DIM T$(200)**, uma vez que a rotina aqui apresentada é muito extensa.

```
10    org 54603
20    ld a,(-5202)
30    cp 0
40    jp nz,pl
50    ld a,(-5203)
60    cp 1
70    jp z,wa
80    ld hl,(62407)
90    ld de,(-5205)
100    add hl,de
110    push hl
120    dec hl
130    ld a,255
140    push hl
150    call 77
160    pop hl
170    ld de,32
180    add hl,de
190    ld a,255
200    call 77
210    pop hl
220    ld a,0
230    push hl
240    call 77
250    pop hl
260    ld de,32
270    add hl,de
280    ld a,1
290    push hl
300    call 77
310    pop hl
320    ld de,32
330    add hl,de
340    call 74
350    cp 72
360    jp z,mre
370    cp 74
380    jp z,mre
390    cp 37
400    jp z,mre
410    cp 254
420    jp z,mre
430    ld a,4
440    call 321
450    bit 2,a
460    jr nz,mf
470    ld b,1
480    bit 3,a
490    jr nz,ml
500    ld b,129
510    ml ld a,b
520    ld (-5202),a
530    jr mc
540    mf bit 3,a
550    jr nz,mc
560    ld a,1
570    ld (-5203),a
580    mc ld hl,(-5205)
590    ld de,222
600    sbc hl,de
610    jr nz,mo
620    ld a,1
630    ld (-5201),a
640    mo ret
650    wa ld hl,(62407)
660    ld de,(-5205)
670    add hl,de
680    ld de,33
690    add hl,de
700    push hl
710    call 74
720    pop hl
730    cp 36
740    jr z,mre
750    cp 52
760    jr z,mt
770    cp 13
780    jr z,mre
790    cp 17
800    jr z,mre
810    ld de,32
820    add hl,de
830    call 74
840    cp 72
850    jr z,mre
860    cp 74
870    jr z,mre
880    cp 254
890    jr z,mre
900    cp 37
910    jr z,mre
920    ld hl,(62407)
930    ld de,(-5205)
940    add hl,de
950    ld a,4
960    push hl
970    call 77
980    pop hl
990    inc hl
1000    ld a,6
1010    push hl
1020    call 77
1030    pop hl
1040    ld de,32
1050    add hl,de
1060    ld a,7
1070    push hl
```

```
1080    call 77
1090    pop hl
1100    dec hl
1110    ld a,5
1120    call 77
1130    ld hl,(-5205)
1140    inc hl
1150    ld (-5205),hl
1160    mt ld a,0
1170    ld ( 5203),a
1180    ret
1190    mre ld a,2
1200    ld (-5201),a
1210    ret
1220    pl ret
1230    end
```

Você pode executar essa rotina mesmo sem as duas restantes, que completam o programa de movimentação de Willie. Embora esta parte possa chamar as demais, não haverá erro. Uma instrução **ret**, que apagaremos mais tarde, foi colocada no fim da rotina — assim, se uma seção inexistente for acessada, o programa simplesmente retornará.

O QUE WILLIE VAI FAZER?

A variável armazenada em −5202 indica se Willie vai ou não pular. O conteudo dessa posição e carregado no acumulador e comparado com 0. Se não for 0, o personagem tem que pular e o processador salta para a rotina **pl**. Esta ainda não foi publicada e, no seu lugar, o processador encontra apenas uma instrução **ret**, voltando para a rotina principal do jogo.

Se Willie não deve pular, ou seja, se o conteúdo de −5202 é 0, o processador continua a execução da rotina. O conteúdo da posição de memória −5203, que guarda a variável da caminhada, é carregado no acumulador.

Há duas figuras para Willie — uma com as pernas juntas e a outra com as pernas abertas. Quando ele está parado, a primeira figura é impressa continuamente no mesmo lugar. Mas, quando está andando, as duas figuras são impressas alternadamente. A variável da caminhada indica ao processador se Willie deve andar uma posição — nesse caso, usa-se a figura de pernas abertas.

A instrução **cp 1** verifica se o personagem deve andar. Se o valor for 1, **jp z,wa** manda o processador para a rotina que faz Willie andar uma posição. Do contrário, o processador continua a executar a mesma rotina.

ANIMAÇÃO

Quando está andando, Willie abre as pernas e dá um passo à frente, aparecendo com as pernas juntas. Temos esta sequência (cada item significa uma chamada na rotina de movimentação):

- Willie é impresso com as pernas juntas porque está imóvel; a tecla N foi pressionada e a variável, ajustada.
- Willie é impresso com as pernas abertas pela rotina que o faz andar; sua posição é incrementada e a variável da caminhada é ajustada com zero no fim da rotina.
- Willie é impresso com as pernas juntas, uma posição à frente, e as teclas são testadas.

Com as pernas abertas, o personagem ocupa duas posições adjacentes; com as pernas juntas, uma. Garantimos assim um efeito de movimentação suave.

Como você pode concluir da sequência acima, toda vez que Willie for impresso com as pernas fechadas, a posição anterior deve ser apagada. Caso contrário, partes da figura serão deixadas pelo caminho.

O par HL é carregado com o endereço inicial da Tabela de Nomes (TN) da VRAM, que se encontra nas posições 62407 e 62408. A posição de Willie na tela, que está em −5205 e −5204, é colocada em DE. Os dois valores são somados em HL, que passa a conter o endereço de Willie na TN da VRAM. Esse endereço é guardado na pilha.

O endereço em HL é decrementado — o que corresponde a voltar uma posição na tela. Esse valor também é guardado na pilha. O acumulador é carregado com 255, o código do padrão vazio usado como céu, e a rotina 77 da ROM é chamada. Como você deve se lembrar, ela escreve o valor do acumulador no endereço da VRAM apontado por HL.

O último valor de HL é recuperado da pilha e adicionado a 32, por DE. Isso faz HL apontar para a posição logo abaixo da anterior. O acumulador é novamente carregado com 255, o padrão de céu, e a rotina 77 é chamada.

Todo esse procedimento teve como finalidade apagar a metade esquerda do nosso personagem que, com as pernas abertas, ocupa quatro posições. A metade direita será apagada quando a figura for impressa com as pernas juntas, sobreposta à anterior. Repare que isso poderia ser desnecessário se Willie já estivesse com as pernas juntas — mas é uma precaução sempre tomada.

IMOBILIDADE

Precisamos agora imprimir a nova figura com as pernas juntas. A posição que foi armazenada na pilha é recuperada em HL. O acumulador é carregado com 0, código do primeiro padrão para essa figura. Guarda-se o valor de HL na pilha e a rotina 77 é chamada.

Ao endereço, recuperado da pilha e colocado em HL, adiciona-se 32 — o que significa apontar uma posição abaixo. O acumulador é carregado com 1, código do segundo padrão da figura. O valor de HL é guardado na pilha e a rotina 77 é chamada. Feito isso, a figura de Willie com as pernas juntas já está impressa.

ONDE WILLIE ESTÁ PISANDO?

Como Willie se moveu uma posição à frente, convém verificar onde ele está pisando. Note que, mesmo que o personagem estivesse parado, esta parte da rotina seria executada. Se ele estiver pisando numa cobra, num buraco ou na água, o processador deve ser informado de sua morte.

A primeira coisa a fazer é verificar qual o código do padrão que está sob os pés de Willie. O endereço que o padrão dos pés ocupa na TN da VRAM é recuperado da pilha e colocado no par HL. O valor 32 é somado a esse endereço por meio do par DE, fazendo com que ele passe a apontar para a posição imediatamente abaixo dos pés de Willie. A rotina 74 da ROM é chamada e faz a leitura da VRAM, devolvendo a A o valor da posição apontada por HL.

Com isso, o acumulador passa a conter o código do padrão no qual Willie está pisando. Esse valor é comparado com 72 e 74, os padrões de mar; com 37, o padrão da cabeça da cobra, e com 254, o padrão do buraco. Se qualquer um desses valores estiver presente, a instrução **jp z,mre** manda o processador para a rotina **mre**, que elimina nosso personagem. Caso contrário, o processador continua executando a rotina — Willie está salvo.

HORA DE SALTAR

A impressão de Willie com as pernas juntas corresponde, efetivamente, ao fim da ação nesta parte da rotina. Desde que o personagem não tenha morrido, deveremos verificar se ele irá se mover na próxima vez. Para isso, examinamos a matriz do teclado.

O teclado do MSX é controlado pela interface de periféricos PPI (8255). Cada tecla faz parte de uma matriz e ocupa uma linha e coluna determinadas. Para verificar se uma tecla está sendo pressionada, mandamos um sinal para

a linha que ela ocupa e checamos se o bit correspondente à sua coluna tem valor 0. Essas operações envolvem escrita e leitura na PPI e podem ser implementadas com instruções **in** e **out** nos endereços adequados. No nosso caso, a rotina 321 da ROM realiza a tarefa. Para isso, basta colocar no acumulador a linha da matriz do teclado que queremos pesquisar. A rotina 321 devolve nos bits do acumulador a situação de todas as teclas dessa linha; se o bit correspondente à tecla for 0, ela está sendo pressionada.

Como as teclas M e N ocupam a quarta linha da matriz do teclado, o acumulador é carregado com 4. Em seguida, a rotina 321 é chamada. A instrução **bit 2,a** analisa o bit dois do número encontrado para verificar se a tecla M está sendo pressionada. Todos os bits correspondentes às teclas têm, normalmente, valor 1. Quando a tecla é pressionada, porém, eles assumem o valor 0. Se a tecla M foi pressionada, o bit dois tem valor 0.

Nesse caso, o processador ignora a instrução **jr nz**. Mas, se M não tiver sido pressionada, ele salta para a rotina **mf**. Como a tecla M foi pressionada, Willie deve pular; o registro B é carregado com 1.

Depois o bit três é testado da mesma maneira. Se N não foi pressionada, o bit três tem valor 1 e a instrução **jr nz** faz o processador saltar a próxima instrução. Mas, se a tecla N foi pressionada, **jr nz** não tem efeito e B é carregado com 129.

Seja como for, o processador transfere o conteúdo de B para A e o coloca em −5202. Esta, como você se lembra, é a posição de memória que o processador examina no início da rotina, para ver se Willie deve ou não pular.

O significado dos números 1 e 129 será explicado nas duas próximas seções de *Avalanche*, que executam o salto vertical e o salto à frente.

A instrução **jr mc** faz o processador pular a próxima parte.

WILLIE DEVE ANDAR?

Se a tecla M não foi pressionada, o processador vai para a rotina **mf**, que checa se a tecla N foi pressionada — ou seja, se Willie não irá pular, mas, simplesmente, andar.

O bit três é testado para ver se a tecla N foi pressionada. Note que a leitura da matriz do teclado ainda se encontra no acumulador — o processador pulou para esta parte da rotina depois que o bit dois foi testado.

Se a tecla N não tiver sido pressionada, a instrução **jr nz** salta a próxima parte. Caso contrário, A é carregado com 1 e esse valor vai para o endereço −5203, onde se encontra a variável da caminhada, examinada no início da rotina. Nessa posição o valor 1 indica que deve ser impressa a figura de Willie que tem as pernas abertas — portanto, ele está andando.

PRÊMIOS

Falta verificar apenas um elemento: Willie pegou o seu prêmio?

A posição do personagem é armazenada nos endereços −5205 e −5204. O conteúdo desses endereços é dado pelo par HL. O valor 222, que corresponde à posição de tela onde o prêmio foi impresso, é colocado no par DE.

O conteúdo de DE é subtraído do conteúdo de HL. Se o resultado for diferente de 0, Willie não alcançou o prêmio. A instrução **jr nz,mo** salta então as próximas instruções. Mas, se o resultado for 0, o personagem já obteve prêmio e o valor 1 é carregado em −5201 através do acumulador. Esta é a posição que a rotina principal examina, informando se o escore deve ser aumentado e outro nível iniciado. Feito isso, o processador encontra a instrução **ret** e retorna.

UM PASSO À FRENTE

Antes de prosseguir, Willie deve verificar o que existe à sua frente. Sua posição é colocada no par DE; o endereço inicial da TN da VRAM, em HL. Os dois valores são somados em HL, que passa a conter o endereço na TN que corresponde à posição da cabeça de Willie na tela. O valor 33 é somado em HL, movendo o apontador uma linha para baixo e uma posição para a direita — em outras palavras, para a posição adiante dos pés de Willie. Esse valor é guardado na pilha e, em seguida, a rotina 74 da ROM é chamada para fazer a leitura da VRAM.

O código do padrão que está na frente dos pés de Willie se encontra no acumulador. Ele é comparado com 36, padrão da língua da cobra; com 13 e 17, padrões da pedra, e com 52, padrão da encosta. Se os padrões da pedra ou da cobra estão presentes, a instrução **jr z,mre** faz o processador saltar para a rotina que elimina Willie. Mas, se na posição estiver o padrão da encosta, o processador salta para a rotina que deixa Willie parado. Obviamente, ele não pode ir em frente a não ser pulando.

Não basta, porém, identificar o padrão que está na frente de Willie. Precisamos saber, também, em que padrão ele pisará depois que se mover. Para isso, DE é carregado com 32 e somado ao conteúdo de HL, que aponta para uma posição abaixo da anterior.

A rotina 74 é novamente chamada e o conteúdo do acumulador comparado com os padrões do mar, do buraco e da língua da cobra. Se qualquer um desses valores estiver presente, a instrução **jr z,mre** vai para a rotina da morte. Se nada disso aconteceu e o personagem ainda está vivo, o par HL é carregado com o endereço inicial da TN, e DE, com a posição de Willie na tela. Esse valor é somado em HL.

A rotina 77 é chamada quatro vezes para imprimir os quatro padrões da figura de Willie com as pernas abertas, cujos códigos são 4, 5, 6 e 7.

A posição do personagem é carregada de −5205 e −5204 para o par HL, incrementada e devolvida. Com isso, atualizamos a posição de Willie. Na próxima vez, ele começará deslocado um caractere para a direita.

AINDA PARADO

A rotina **mt** é chamada diretamente quando Willie se depara com a encosta, não podendo prosseguir a não ser com um pulo. O processador, porém, chega a essa rotina quando o personagem foi impresso com as pernas abertas.

Um valor 0 é carregado no endereço −5203, por intermédio do acumulador, indicando ao processador que, na próxima vez, a figura de Willie deve ser impressa com as pernas juntas, no mesmo lugar ou uma posição à frente, dependendo do que aconteceu.

Em seguida, o processador retorna.

O ÚLTIMO SUSPIRO

Se Willie se afogou no mar, foi picado por uma cobra, caiu num buraco ou foi atingido por uma pedra, a rotina **mre** é chamada, informando ao resto do programa que Willie morreu.

Para que se divulgue essa informação, o valor 2 é colocado no endereço −5201, que será examinado em algum ponto do jogo. As providências para o funeral de Willie serão, então, tomadas.

Feito isso, o processador retorna.

O último **ret** da listagem, precedido do rótulo **pl**, será apagado pela próxima rotina de *Avalanche*. Sua única função é evitar que, na falta das rotinas, ocorra algum erro.

UMA PLANILHA ELETRÔNICA (3)

- COMO DAR ENTRADA ÀS EQUAÇÕES
- CÓPIA ABSOLUTA E RELATIVA
- O USO DE CONSTANTES
- COMPLETANDO O PROGRAMA

Com este artigo, encerramos nossa série sobre planilhas eletrônicas. Depois de acrescentar ao programa as linhas aqui apresentadas, você poderá explorar todo o seu potencial.

Para ter uma planilha completa e operacional, carregue na memória do seu micro as partes já dadas do programa e acrescente as linhas que faltam. Nos artigos anteriores, mostramos, por meio de exemplos, como montar uma planilha. Agora, que você já tem todo o programa, veremos como utilizá-lo.

As instruções que se seguem são válidas para todas as máquinas. As diferenças existentes entre o Spectrum e os demais microcomputadores serão sempre especificadas.

MONTAGEM DAS EQUAÇÕES

O que faz da planilha eletrônica uma ferramenta tão versátil é sua capacidade de trabalhar com equações. Não se assuste com a palavra equação: nossas "equações" não passam de especificações bastante simples das operações aritméticas de adicionar valores das colunas, obter o total de uma determinada linha, calcular porcentagem etc. Os ope-



MICRO DICAS

USE A PLANILHA COM MAIS EFICIÊNCIA

Agora que você dispõe de um programa para a elaboração de uma planilha de cálculo, convém conhecer alguns "truques" que lhe permitirão utilizá-la de maneira bem mais rápida e eficiente.

• Preenchimento da planilha: depois de definir o título geral e os subtítulos das seções, dedique-se ao "esqueleto" da planilha, ou seja, à identificação das linhas e colunas. Em seguida, coloque os dados propriamente ditos e, finalmente, as fórmulas. Procedendo assim, você diminuirá as chances de cometer erros de entrada e evitará que se executem cálculos antes que todos os dados e rótulos estejam presentes.

• Cuidado com os laços sem fim: esse tipo de problema pode ocorrer quando se colocam duas fórmulas em células diferentes, uma utilizando o resultado do cálculo da outra. Em conseqüência, o programa "emperra", pois a detecção de uma situação como esta não foi prevista. Não deixe de verificar todas as fórmulas antes de executar o programa.

radores usados são: mais (+), menos (−), vezes (*), dividido (/), por cento (%), total de linhas (&) e total de colunas ($).

Para escrever uma equação, indica-se o nome da primeira célula, depois o nome da segunda célula e, por último, o operador. Vejamos alguns exemplos. Se colocarmos A1A2+ na célula C1, obteremos a soma dos valores contidos em A1 e A2 na célula C1. A equação A1A10$ soma todos os valores da coluna A, da linha 1 até a linha 10. Da mesma maneira, A6F6& soma todos os valores da linha 6, começando na coluna A e terminando na coluna F. Finalmente, A5B2% é calculado como A5*B2/100, o que dá B2 por cento de A5. As equações são colocadas nas células em que se quer que apareçam os resultados.

Com exceção das versões destinadas ao Spectrum e ao Apple, o programa permite ainda que se acrescente um número no fim da equação para especificar a quantidade de casas decimais que aparecem na resposta.

CÓPIA ABSOLUTA E RELATIVA

O programa oferece a alternativa de se copiar o conteúdo de uma célula. Isso nos poupa o trabalho de redigitar uma fórmula em todas as células em que será usada. Podemos escolher entre uma cópia absoluta ou relativa. A cópia absoluta produz uma réplica idêntica da célula, onde quer que você queira, seja numa única célula, seja numa coluna inteira. A cópia relativa, usada especialmente para equações, sofre alterações conforme a linha e a coluna para onde está sendo copiada.

Suponhamos que você tem A1B1* na célula C1 e quer que toda célula da coluna C contenha o produto das duas colunas anteriores — ou seja, A2B2* em C2, A3B3* em C3 etc. A cópia relativa faz isso para você. O procedimento é o seguinte: pressione a tecla de cópia (verifique, adiante, qual a tecla que seu micro usa), selecione R (para relativa) e coloque o cursor sobre a célula a ser copiada (no Spectrum, digita-se o nome da célula). Em seguida, defina se a cópia deve ser feita em linha ou coluna — no nosso caso, pressione C. Finalmente, digite o nome da célula inicial (C1) e da final (C10, por exemplo). O programa, assim, preencherá toda a coluna para você.

Experimente também copiar valores em linhas e obter cópias absolutas de equações e valores.

CONSTANTES

Precisamos, com freqüência, utilizar constantes em uma equação — por exemplo, um valor como 17, para calcular a porcentagem do ICM, ou como 10, para calcular descontos.

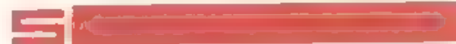
Como não se pode colocar números diretamente em uma equação — que é baseada no conteúdo de duas células —, empregam-se diferentes métodos para a introdução de constantes.

No Spectrum, elas costumam ser precedidas pela letra Z. Assim, A1Z17% calcula o ICM de um valor colocado na célula A1. Nos demais micros, as constantes devem ser colocadas na coluna Z. Retomando o último exemplo: o valor 17 será colocado na célula Z1 e a equação tomará a forma A1Z1%.

Esse procedimento é necessário porque os valores das constantes não devem ser alterados nem mesmo quando se opta por uma cópia relativa. Se reservarmos uma coluna para elas, estaremos assegurando um tratamento diferenciado na hora da cópia.

O USO DO PROGRAMA

Existem algumas variações, de máquina para máquina, no que diz respeito à colocação de valores, rótulos e equações na planilha, assim como à execução das operações. Comentaremos, em seguida, os detalhes específicos para cada microcomputador.



O programa do Spectrum demora um pouco para definir as variáveis internas. Assim, você terá uma tela vazia por alguns momentos após teclar **RUN**. A tela aparece no modo de valores — o que significa que valores e rótulos podem ser inseridos. Pressione E para passar ao modo de equação e V para voltar ao de valores. Para introduzir um dado em uma célula, use as teclas de controle do cursor; ao chegar à célula correta, pressione I. Em seguida, digite o que quiser.

As equações são mostradas sobre fundo amarelo, com os valores em preto e os rótulos em azul. Considera-se rótulo qualquer dado digitado no modo de valores que comece com uma letra. Como as cores aparecem em qualquer modo, você pode distinguir facilmente as células que contêm equações mesmo que esteja vendo a folha no modo de valores ou vice-versa.

A tela exibe apenas quatro colunas e dez linhas de cada vez, mas é possível mudar para outra parte da planilha pressionando-se simultaneamente as teclas de controle do cursor e **<SYMBOL SHIFT>**

Tente inserir alguns valores e rótulos, mude para o modo de equações e some os valores das colunas e linhas. Para ver o resultado dos cálculos, volte para a tela de valores e pressione C e **<SYMBOL SHIFT>**. Se algum número for muito grande e não couber na célula, os dígitos da direita serão eliminados e os que permanecerem na tela ficarão piscando como aviso.

Agora, experimente copiar algumas equações e valores usando a tecla Z. O programa pedirá toda a informação que for necessária.

Segue-se a lista de comandos usados pelo programa:

V mostra a tela de valores;
E mostra a tela de equações;
I insere um dado em uma célula;
I seguido de **<ENTER>** apaga um valor ou rótulo;
I seguido de # apaga uma equação;
C mais **<SYMBOL SHIFT>** calcula os valores da planilha;
<SPACE> mais **<SYMBOL SHIFT>**

interrompe os cálculos;
Z copia uma célula;
S mais <SYMBOL SHIFT> grava dados em fita;
J mais <SYMBOL SHIFT> carrega dados da fita;
P imprime a tela em uma impressora;
Teclas de controle do cursor movem o cursor;
Teclas de controle mais <SYMBOL SHIFT> movem a janela de texto.

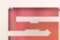


Ao executar o programa, você verá apenas a parte superior esquerda da planilha; se quiser examinar qualquer outra parte, recorra às teclas de controle do cursor (no micro Apple, mova o cursor para cima e para baixo usando as teclas A e Z). Você pode também passar diretamente para qualquer ponto da planilha digitando G.

As teclas V e E alternam as telas de equações e valores. É possível entrar valores nas duas telas, mas os resultados só serão mostrados na tela de valores. Para inserir um dado em uma célula, mova o cursor até ela e pressione I. Depois, digite sua entrada.

As equações são digitadas conforme descrevemos anteriormente. No TRS-Color e no MSX podemos discriminar o número de casas decimais do resultado. A1B1/2, por exemplo, mostra o resultado de A1 dividido por B1 com duas casas decimais. Esse recurso é muito útil quando estamos lidando com valores financeiros. No caso de obtermos números que excedam o tamanho da célula, a mensagem <OV> será exibida.

Eis as teclas usadas no programa:
V mostra a tela de valores;
E mostra a tela de equações;
I insere um rótulo, valor ou equação;
C copia uma célula;
S grava dados no cassete ou disco;
L lê dados do cassete ou disco;
Q sai do programa;
Teclas de controle do cursor movem o cursor e a janela de texto (no Apple, usam-se A e Z para mover o cursor para cima e para baixo).



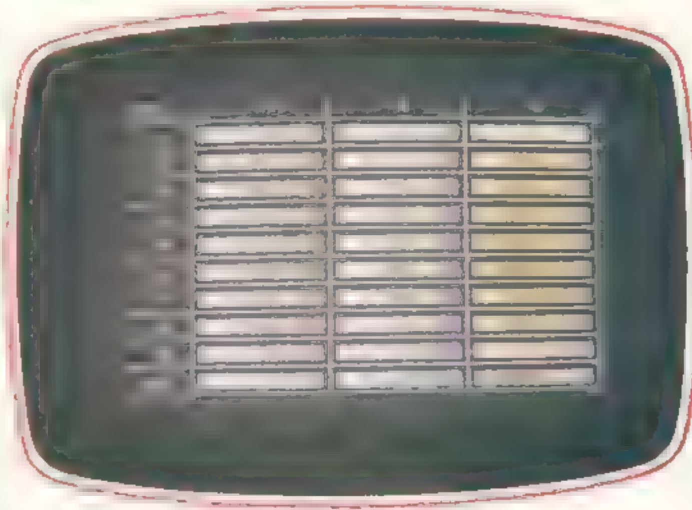
```
910 IF z=3 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2): LET v(3)=(CODE s$(3))-64
: LET v(4)=VAL s$(4 TO 5): LET
o$=s$(6): RETURN
920 IF z=4 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2 TO 3): LET v(3)=(CODE s$(4
))-64: LET v(4)=VAL s$(5 TO 6)
: LET o$=s$(7): RETURN
930 IF z=5 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2): LET v(3)=(CODE s$(3))-64
: LET v(4)=VAL s$(4): LET o$=s
$(5): RETURN
940 IF z=6 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2 TO 3): LET v(3)=(CODE s$(4
))-64: LET v(4)=VAL s$(5): LET
o$=s$(6): RETURN
950 IF z=7 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2): LET v(3)=(CODE s$(3))-64
: LET v(4)=VAL s$(4 TO 5): LET
o$=s$(6): RETURN
960 IF z=8 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2 TO 3): LET v(3)=(CODE s$(4
))-64: LET v(4)=VAL s$(5 TO 6)
: LET o$=s$(7): RETURN
970 IF z=9 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2): LET v(3)=(CODE s$(3))-64
: LET v(4)=VAL s$(4 TO 6): LET
o$=s$(7): RETURN
980 IF z=10 THEN  LET v(1)=(
CODE s$(1))-64: LET v(2)=VAL s
$(2 TO 3): LET v(3)=(CODE s$(4
))-64: LET v(4)=VAL s$(5 TO 7
): LET o$=s$(8): RETURN
990 LET v(1)=(CODE s$(1))-64:
LET v(2)=VAL s$(2): LET v(3)=(
CODE s$(3))-64: LET v(4)=VAL s
$(4 TO 7): LET o$=s$(8):
RETURN
1000 RETURN
1010 IF d$(v(2),v(1),1)>"9" THE
N  LET t=0: RETURN
1020 IF v(3)=26 THEN  LET b=v(4
): LET a=VAL d$(v(2),v(1), TO 8
): GOTO 1050
1030 IF d$(v(4),v(3),1)>"9" THE
N  LET t=0: RETURN
1040 LET a=VAL d$(v(2),v(1), TO
 8): LET b=VAL d$(v(4),v(3), TO
 8)
1050 IF o$="+" THEN  LET t=a+b
 RETURN
1060 IF o$="-" THEN  LET t=a-b
 RETURN
1070 IF o$="/" AND b=0 THEN  LE
T t=0: RETURN
1080 IF o$="/" THEN  LET t=a/b:
 RETURN
1090 IF o$="*" THEN  LET t=a*b:
 RETURN
1100 IF o$="%" THEN  LET t=(a*b
)/100: RETURN
1110 IF o$="$" THEN  LET t=0: F
OR s=v(2) TO v(4): LET t=t+VAL
d$(s,v(1), TO 8): NEXT s: RETUR
N
1120 IF o$="&" THEN  LET t=0: F
OR s=v(1) TO v(3): LET t=t+VAL
d$(v(2),s, TO 8): NEXT s: RETUR
N
1130 RETURN
1140 IF z=1 THEN  IF VAL a$(2)=
0 OR VAL a$(4)=0 THEN  LET f=1:
 RETURN
1150 IF z=2 THEN  IF VAL a$(2 T
O 3)>30 OR VAL a$(2 TO 3)=0 OR
VAL a$(5)=0 THEN  LET f=1: RETU
RN
1160 IF z=3 THEN  IF VAL a$(4 T
O 5)>30 OR VAL a$(4 TO 5)=0 OR
VAL a$(2)=0 THEN  LET f=1: RETU
RN
1170 IF z=4 THEN  IF VAL a$(2 T
O 3)>30 OR VAL a$(5 TO 6)>30 OR
 VAL a$(2 TO 3)=0 OR VAL a$(5 T
O 6)=0 THEN  LET f=1: RETURN
1180 IF z=5 OR z=7 OR z=11 THEN
  IF VAL a$(2)=0 THEN  LET f=1:
 RETURN
1190 IF z=6 OR z=8 OR z=10 THEN
  IF VAL a$(2 TO 3)=0 OR VAL a$
(2 TO 3)>30 THEN  LET f=1: RETU
RN
1200 LET s$="          ": FOR g=1
 TO c: LET s$(g)=a$(g): NEXT g:
 GOSUB 890: IF o$="&" AND (v(1)
>v(3) OR v(2)<>v(4)) THEN  LET
f=1: RETURN
1210 IF o$="$" AND (v(1)<>v(3)
OR v(2)>v(4)) THEN  LET f=1: RE
TURN
1220 LET f=0: RETURN
1230 LET g$=d$(y+wy,x+wx,17)
1240 PRINT  FLASH 0+(g$="5" AND
 t$="VAL"): INK 0+(2 AND g$="4"
): PAPER 7-(2 AND g$="2")-(g$="
1" OR g$="5");AT y*2+2,x*9+5;(d
$(y+wy,x+wx, TO 8) AND t$="VAL"
);(d$(y+wy,x+wx,9 TO 16) AND t$
="IGUA"): RETURN
1250 SOUND .4,10
1260 LET s$="          ": LET os=
0
1270 PAUSE 20: PRINT AT cy*2,(c
x-1)*9+5+os; BRIGHT 1; OVER 1;"
"
1280 IF CODE INKEY$>30 AND CODE
 INKEY$<128 THEN  LET s$(os+1)=
INKEY$: PRINT AT cy*2,(cx-1)*9+
5+os;s$(os+1): LET os=os+1: GOT
O 1350
1290 IF CODE INKEY$=12 AND os>0
 THEN  LET os=os-1: LET s$(os+1
)="": PRINT AT cy*2,((cx-1)*9)+
5+os; OVER 0; BRIGHT 0;" ": GO
TO 1270
1300 IF CODE INKEY$=13 AND s$="
        " THEN  RETURN
1310 IF CODE INKEY$=13 AND t$="
VAL" AND CODE s$(1)>64 THEN  LE
T d$(wy+cy-1,wx+cx-1,17)="2": L
ET d$(wy+cy-1,wx+cx-1, TO 8)=s$
: LET y=cy-1: LET x=cx-1: GOSUB
 1230: RETURN
1320 IF CODE INKEY$=13 AND t$="
VAL " THEN  GOSUB 1380: LET os=
0: LET d$(wy+cy-1,wx+cx-1, TO 8
)=s$: LET y=cy-1: LET x=cx-1: G
OSUB 1230: RETURN
1330 IF CODE INKEY$=13 AND t$="
IGUA" THEN  GOSUB 1380: LET os=
0: LET d$(wy+cy-1,wx+cx-1,9 TO
16)=s$: LET x=cx-1: LET y=cy-1:
 GOSUB 1230: RETURN
1340 GOTO 1270
1350 IF os=8 AND t$="VAL" THEN
 SOUND .1,-10: GOSUB 1380: LET
d$(wy+cy-1,wx+cx-1, TO 8)=s$: L
```

```
ET y=cy-1: LET x=cx-1: GOSUB 12
30: RETURN
1360 IF os=8 AND t$="IGUA" THEN
 SOUND .1,+10: GOSUB 1380: LET
 d$(wy+cy-1,wx+cx-1,9 TO 16)=s$
 LET y=cy-1: LET x=cx-1: GOSUB
 1230· RETURN
1370 GOTO 1270
1380 PRINT AT cy*2,((cx-1)*9)+5
+os; BRIGHT 0; OVER 1;" "
1390 IF t$="IGUA" THEN  GOSUB 1
490: RETURN
1400 GOSUB 1410: PRINT AT cy*2,
(cx-1)*9+5;s$: RETURN
1410 LET b$="            000": FOR
u=1 TO os: LET b$(u+(8-os))=s$(
u): NEXT u
1420 FOR u=1 TO 11
1430 IF b$(u)<>"." THEN  NEXT u
1440 IF u>=11 THEN  LET b$(9)="
.": LET s$=b$(4 TO ): RETURN
1450 LET w=6-u
1460 IF w<0 THEN  LET w=ABS w+1
: LET s$=b$(w TO w+8): RETURN
1470 LET s$="          "
1480 FOR u=1 TO 8-w: LET s$(w+u
)=b$(u): NEXT u: RETURN
1490 IF s$(1)="#" THEN  LET s$=
"          "· LET d$(wy+cy-1,wx+c
x-1,17)="0": RETURN
1500 LET a$="". FOR z=1 TO 8· L
ET a$=a$+(s$(z) AND s$(z)<>" ")
1510 NEXT z: LET c=LEN a$
1520 IF c<4 THEN  LET d$(wy+cy-
1,wx+cx-1,17)="4": RETURN
1530 RESTORE 1630: FOR z=1 TO 1
1: LET f=0· READ m$
1540 FOR w=1 TO c
1550 IF m$(w)="A" THEN  GOSUB 1
650: IF f THEN  GOTO 1610
1560 IF m$(w)="N" THEN  GOSUB 1
670: IF f THEN  GOTO 1610
1570 IF m$(w)="O" THEN  GOSUB 1
690· IF f THEN  GOTO 1610
1580 IF m$(w)="Z" THEN  GOSUB 1
710· IF f THEN  GOTO 1610
1590 NEXT w: GOSUB 1140: IF NOT
 f THEN  LET s$="           ": FOR
 w=1 TO c: LET s$(w)=a$(w): NEX
T w: LET d$(wy+cy-1,wx-cx-1,18)
=CHR$ z: LET d$(wy+cy-1,wx+cx-1
,17)="1"· RETURN
1600 GOTO 1620
1610 NEXT z
1620 LET d$(wy+cy-1,wx+cx 1,17)
="4": RETURN
1630 DATA "ANANOOO","ANNANOO","
ANANNOO","ANNANNO"
1640 DATA "ANZNOOOO","ANNZNOOO"
,"ANZNNOOO","ANNZNNOO","ANZNNNO
O","ANNZNNNO","ANZNNNNO"
1650 IF a$(w)>="A" AND a$(w)<="
X" THEN  RETURN
1660 LET f=1· RETURN
1670 IF a$(w)>="0" AND a$(w)<="
9" THEN  RETURN
1680 LET f=1  RETURN
1690 LET c$=a$(w)· IF c$="+" OR
 c$="-" OR c$="*" OR c$="/" OR
c$="£" OR c$="$" OR c$="&" THEN
 RETURN
1700 LET f=1: RETURN
1710 IF a$(w)="Z" THEN  RETURN
1720 LET f=1: RETURN
1730 FOR y=1 TO 30: FOR x=1 TO
24: LET d$(y,x, TO 8)="    0.00
"· LET d$(y,x,17) "0": LET d$(y
,x,18)=CHR$ 0: NEXT x: NEXT y
1740 RESTORE 1750  FOR x=USR "a
" TO USR "c"+7: READ a: POKE x,
a: NEXT x: RETURN
1750 DATA 8,8,8,8,8,8,8,8
1760 DATA 0,0,0,255,0,0,0,0
1770 DATA 8,8,8,255,8,8,8,8
1780 INPUT "NOME DO ARQUIVO ?";
n$
1790 SAVE n$ DATA d$()
1800 PRINT #1;AT 0,0;"PRESSIONE
 V PARA VERIFICAR "
1810 PAUSE 0· IF INKEY$ "V" OR
INKEY$="v" THEN  GOTO 1830
1820 RETURN
1830 VERIFY n$ DATA d$()· RETUR
N
1840 INPUT "NOME LO ARQUIVO ?",
n$
1850 LOAD n$ DATA d$(): RETURN
```

```
1490 PRINT @448:PRINT @448,;:IF
 LEN(DS(CC,CR))=1 THEN PRINT"NA
DA A COPIAR".SOUND 5,5 FOR K=1
TO 500:NEXT:RETURN ELSE PRINT "
COPIA aBSOLUTA OU rELATIVA";
1500 A$=INKEY$:IF A$="" THEN 15
00
1510 IF A$=CHR$(13) THEN RETURN
1520 IF A$="A" THEN TC=1:GOTO 1
540
1530 IF A$<>"R" THEN 1490 ELSE
TC=0
1540 PRINT @448
1550 IF TC=0 AND ASC(DS(CC,CR))
<>131 THEN PRINT @448,"ERRADO :
 MODO COPIA - SOMENTE   EQUACOE
S";:SOUND 1,5 FOR D=1 TO 500·NE
XT D GOTO 490
1560 PRINT @448:PRINT @448,"COP
IA DE lINHA OU cOLUNA";
1570 A$=INKEY$:IF A$="" THEN 15
70
1580 IF A$=CHR$(13) THEN RETURN
1590 IF A$="C" THEN DC=1.GOTO 1
610
1600 IF A$<>"L" THEN 1560 ELSE
DC=0
1610 PRINT @448·PRINT @448,"COM
ECAR PELA CELULA";:INPUT A$:IF
A$="" THEN RETURN
1620 S1=ASC(A$)-64:IF S1<1 OR S
1>25 THEN 1610
1630 S2=VAL(MID$(A$,2)):IF S2<1
 OR S2>30 THEN 1610
1640 PRINT @448:PRINT @448,"TER
MINAR NA CELULA";:INPUT A$
1650 IF A$="" THEN RETURN
1660 C1=ASC(A$)-64.IF C1<1 OR C
1>25 THEN 1640
1670 C2=VAL(MID$(A$,2)):IF C2<1
 OR C2>30 THEN 1640
1680 PRINT @448,"COPIANDO  -  E
SPERE"
```



Entrada das equações no Spectrum.

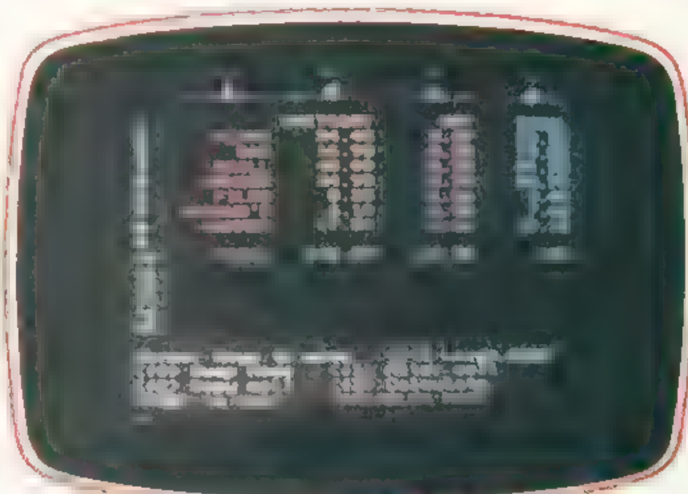


Cálculo dos resultados na tela do Spectrum.

```
1690 IF TC=1 THEN 1970
1700 IF DC=1 THEN 1840
1710 IF C2<>S2 THEN 2090
1720 AS=ASC(MID$(D$(CC,CR),2))-
64:IF AS<>26 AND AS+C1-S1>25 TH
EN C1=25+S1-AS
1730 AS=ASC(MID$(D$(CC,CR),5))-
64·IF AS<>26 AND AS+C1-S1>25 TH
EN C1=25+S1-AS
1740 D$(S1,S2)=D$(CC,CR)
1750 FOR I=S1+1 TO C1
1760 A$=D$(I-1,S2)
1770 IF MID$(A$,2,1)="Z" THEN 1
790
1780 MID$(A$,2,1)=CHR$(ASC(MID$
(A$,2,1))+1)
1790 IF MID$(A$,5,1)="Z" THEN 1
810
1800 MID$(A$,5,1)=CHR$(ASC(MID$
(A$,5,1))+1)
1810 D$(I,S2)=A$·NEXT
1820 IF C1>CX THEN CX=C1
1830 GOTO 2080
1840 IF C1<>S1 THEN 2090
1850 AS=VAL(MID$(D$(CC,CR),3,2)
) IF AS+C2-S2>30 THEN C2=30+S2-
AS
1860 AS=VAL(MID$(D$(CC,CR),6)).
IF AS+C2-S2>30 THEN C2=30+S2-AS
1870 D$(S1,S2)=D$(CC,CR)
1880 FOR I=S2+1 TO C2
1890 A$=D$(S1,I-1)
1900 IF MID$(A$,2,1)="Z" THEN 1
920
1910 V=VAL(MID$(A$,3,2))+1:IF V
<10 THEN MID$(A$,3,2)=STR$(V) E
LSE MID$(A$,3,2)=MID$(STR$(V),2
,2)
1920 IF MID$(A$,5,1)="Z" THEN 1
940
1930 V=VAL(MID$(A$,6,2))+1.IF V
<10 THEN MID$(A$,6,2)=STR$(V) E
LSE MID$(A$,6,2)=MID$(STR$(V),2
,2)
1940 D$(S1,I)=A$:NEXT
1950 IF C2>RX THEN RX=C2
1960 GOTO 2080
1970 IF DC=1 THEN 2030
1980 IF C2<>CR THEN 2090
1990 FOR I=CC+1 TO C1
2000 D$(I,CR)=D$(I-1,CR):NEXT
2010 IF C1>CX THEN CX=C1
2020 GOTO 2080
2030 IF C1<>CC THEN 2090
2040 FOR I=CR+1 TO C2
2050 D$(CC,I)=D$(CC,I-1):NEXT
2060 IF C2>RX THEN RX=C2
2070 GOTO 2080
2080 MO=1·GOSUB 70·RETURN
2090 PRINT @448,"erro NO DESTIN
O":SOUND 1,5:FOR D=1 TO 500:NEX
T D
2100 GOTO 2080
```

```
1490 LOCATE 0,21:PRINTSPC(35)·L
OCATE 0,21:IF LEN(D$(CC,CR))=1
THEN PRINT"NADA PARA COPIAR"·FO
R K=1 TO 500·NEXT·RETURN ELSE P
RINT"CÓPIA <A>BSOLUTA OU <R>ELA
TIVA: ";:
1500 A$=INKEY$·IF A$="" THEN 15
00
1510 IF A$=CHR$(13) THEN RETURN
1520 IF A$="A" THEN TC=1·GOTO 1
540
1530 IF A$<>"R" THEN 1490 ELSE
TC=0
1540 LOCATE 0,21·PRINTSPC(35):L
OCATE 0,21
1550 IF TC=0 AND ASC(D$(CC,CR))
<>131 THEN PRINT"MODO DE CÓPIA
ERRADO - SÓ EQUAÇÕES!";:FOR D=1
 TO 500·NEXT·GOTO 490
1560 LOCATE 0,21·PRINT"CÓPIA PO
R <LINHA> OU <COLUNA>:     ";
1570 A$=INKEY$:IF A$="" THEN 15
70
1580 IF A$=CHR$(13) THEN RETURN
1590 IF A$="C" THEN DC=1:GOTO 1
610
1600 IF A$<>"L" THEN 1570 ELSE
DC=0
1610 LOCATE 0,21·PRINTSPC(37):L
OCATE 0,21:PRINT"INICIA NA CÉLU
LA";:INPUT A$:IF A$="" THEN RET
URN
1620 S1=ASC(A$)-64:IF S1<1 OR S
1>25 THEN 1610
1630 S2=VAL(MID$(A$,2)):IF S2<1
 OR S2>30 THEN 1610
1640 LOCATE 0,21·PRINTSPC(37):L
OCATE 0,21:PRINT"TERMINA NA CÉL
ULA";:INPUT A$
1650 IF A$="" THEN RETURN
1660 C1=ASC(A$)-64:IF C1<1 OR C
1>25 THEN 1640
1670 C2=VAL(MID$(A$,2))·IF C2<1
 OR C2>30 THEN 1640
1680 LOCATE 0,21·PRINTSPC(35)·L
OCATE 0,21:PRINT"COPIANDO... AG
UARDE"
1690 IF TC=1 THEN 1970
1700 IF DC=1 THEN 1840
1710 IF C2<>S2 THEN 2090
1720 AW=ASC(MID$(D$(CC,CR),2))-
64:IF AW<>26 AND AW+C1-S1>25 TH
EN C1=25+S1-AW
1730 AW=ASC(MID$(D$(CC,CR),5))-
64·IF AW<>26 AND AW+C1-S1>25 TH
EN C1=25+S1-AW
1740 D$(S1,S2)=D$(CC,CR)
1750 FOR I=S1+1 TO C1
1760 A$=D$(I-1,S2)
1770 IF MID$(A$,2,1)="Z" THEN 1
790
1780 MID$(A$,2,1)=CHR$(ASC(MID$
(A$,2,1))+1)
1790 IF MID$(A$,5,1)="Z" THEN 1
810
1800 MID$(A$,5,1)=CHR$(ASC(MID$
(A$,5,1))+1)
1810 D$(I,S2)=A$:NEXT
1820 IF C1>CX THEN CX=C1
1830 GOTO 2080
1840 IF C1<>S1 THEN 2090
1850 AW=VAL(MID$(D$(CC,CR),3,2)
):IF AW+C2-S2>30 THEN C2=30+S2-
AW
1860 AW=VAL(MID$(D$(CC,CR),6)):
IF AW+C2-S2>30 THEN C2=30+S2-AW
1870 D$(S1,S2)=D$(CC,CR)
```



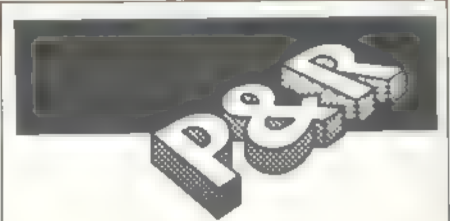


Tela de equações (à esquerda) e entrada de valores e rótulos no Acorn, micro sem equivalente no Brasil.

```
1880 FOR I=S2+1 TO C2
1890 AS=DS(S1,I-1)
1900 IF MIDS(AS,2,1)="Z" THEN 1
920
1910 V=VAL(MIDS(AS,3,2))+1·IF V
<10 THEN MIDS(AS,3,2)=STRS(V) E
LSE MIDS(AS,3,2)=MIDS(STRS(V),2
,2)
1920 IF MIDS(AS,5,1)="Z" THEN 1
940
1930 V=VAL(MIDS(AS,6,2))+1·IF V
<10 THEN MIDS(AS,6,2)=STRS(V) E
LSE MIDS(AS,6,2)=MIDS(STRS(V),2
,2)
1940 DS(S1,I)=AS:NEXT
1950 IF C2>RX THEN RX=C2
1960 GOTO 2080
1970 IF DC=1 THEN 2030
1980 IF C2<>CR THEN 2090
1990 FOR I=CC+1 TO C1
2000 DS(I,CR)=DS(I-1,CR)·NEXT
2010 IF C1>CX THEN CX=C1
2020 GOTO 2080
2030 IF C1<>CC THEN 2090
2040 FOR I=CR+1 TO C2
2050 DS(CC,I)=DS(CC,I-1):NEXT
2060 IF C2>RX THEN RX=C2
2070 GOTO 2080
2080 MO=1 GOSUB 70·RETURN
2090 LOCATE 0,21 PRINTSPC(35)·L
OCATE 0,21:PRINT"ERRO NA DESTIN
AÇAO":FOR D=1 TO 500:NEXTD
2100 GOTO 2080
```



```
1490  VTAB 23: HTAB 1: CALL  -
958· IF  LEN (DS(CC,CR)) = 1 TH
EN  PRINT "NADA PARA COPIAR"; C
HRS (7);: FOR K = 1 TO 1000· NE
XT : RETURN
1495  PRINT "COPIA [A]BSOLUTA O
U [R]ELATIVA ";
1500  GET AS
1510  IF AS =  CHRS (13) THEN 2
085
1520  IF AS = "A" THEN TC = 1·
GOTO 1540
1530  IF AS <  > "R" THEN 1490
1535 TC = 0
1540  VTAB 23: HTAB 1: CALL  -
958
1550  IF TC = 0 AND  ASC (DS(CC
,CR)) <  > 13 THEN  PRINT "MOD
O DE COPIA INCORRETO - SO EQUAC
OES"; CHRS (7);· FOR D = 1 TO 1
000: NEXT : GOTO 490
1560  VTAB 23: HTAB 1: CALL  -
958. PRINT "COPIA [C]OLUNA OU [
L]INHA ";
1570  GET AS
1580  IF AS =  CHRS (13) THEN 2
085
1590  IF AS = "C" THEN DC = 1:
GOTO 1610
1600  IF AS <  > "L" THEN 1560
1605 DC = 0
1610  VTAB 23: HTAB 1: CALL  -
958: PRINT "COMECA NA CEL.: ";:
 INPUT AS: IF AS = "" THEN 2085
1620 S1 =  ASC (AS) - 64: IF S1
 < 1 OR S1 > 25 THEN 1610
1630 S2 =  VAL ( MIDS (AS,2)):
IF S2 < 1 OR S2 > 30 THEN 1610
1640  VTAB 23: HTAB 1: CALL  -
958: PRINT "TERMINA NA CEL.: ";
: INPUT AS
1650  IF AS = "" THEN 2085
1660 C1 =  ASC (AS) - 64: IF C1
 < 1 OR C1 > 25 THEN 1640
1670 C2 =  VAL ( MIDS (AS,2)):
IF C2 < 1 OR C2 > 30 THEN 1640
1680  VTAB 23: HTAB 1: CALL  -
958: PRINT "COPIANDO - AGUARDE.
. ";
1690  IF TC = 1 THEN 1970
1700  IF DC = 1 THEN 1840
1710  IF C2 <  > S2 THEN 2090
1720 AS =  ASC ( MIDS (DS(CC,CR
),2)) - 64: IF AS <  > 26 AND A
S + C1 - S1 > 25 THEN C1 = 25 +
 S1 - AS
1730 AS =  ASC ( MIDS (DS(CC,CR
),5)) - 64: IF AS <  > 26 AND A
S + C1 - S1 > 25 THEN C1 = 25 +
 S1 - AS
1740 DS(S1,S2) = DS(CC,CR)
1750  FOR I = S1 + 1 TO C1
1760 AS = DS(I - 1,S2)
1770  IF  MIDS (AS,2,1) = "Z" T
HEN 1790
1780 AS =  LEFTS (AS,1) +  CHRS
 ( ASC ( MIDS (AS,2,1)) + 1) +
 MIDS (AS,3)
1790  IF  MIDS (AS,5,1) = "Z" T
HEN 1810
1800 AS =  LEFTS (AS,4) +  CHRS
 ( ASC ( MIDS (AS,5,1)) + 1) +
 MIDS (AS,6)
1810 DS(I,S2) = AS· NEXT
1820  IF C1 > CX THEN CX = C1
1830  GOTO 2080
1840  IF C1 <  > S1 THEN 2090
1850 AS =  VAL ( MIDS (DS(CC,CR
),3,2)): IF AS + C2 - S2 > 30 T
HEN C2 = 30 + S2 - AS
1860 AS =  VAL ( MIDS (DS(CC,CR
),6)): IF AS + C2 - S2 > 30 THE
N C2 = 30 + S2 - AS
1870 DS(S1,S2) = DS(CC,CR)
1880  FOR I = S2 + 1 TO C2
1890 AS = DS(S1,I - 1)
1900  IF  MIDS (AS,2,1) = "Z" T
HEN 1920
1910 V =  VAL ( MIDS (AS,3,2))
+ 1: IF V < 10 THEN AS =  LEFTS
 (AS,2) + " " +  STRS (V) +  MI
DS (AS,5): GOTO 1920
1915 AS =  LEFTS (AS,2) +  STRS
 (V) +  MIDS (AS,5)
1920  IF  MIDS (AS,5,1) = "Z" T
HEN 1940
1930 V =  VAL ( MIDS (AS,6,2))
+ 1: IF V < 10 THEN AS =  LEFTS
 (AS,5) + " " +  STRS (V) +  MI
DS (AS,8)· GOTO 1920
1935 AS =  LEFTS (AS,5) +  STRS
 (V) +  MIDS (AS,8)
1940 DS(S1,I) = AS: NEXT
1950  IF C2 > RX THEN RX = C2.
1960  GOTO 2080
1970  IF DC = 1 THEN 2030
1980  IF C2 <  > CR THEN 2090
1990  FOR I = CC + 1 TO C1
2000 DS(I,CR) = DS(I - 1,CR): N
EXT
2010  IF C1 > CX THEN CX = C1
2020  GOTO 2080
2030  IF C1 <  > CC THEN 2090
2040  FOR I = CR + 1 TO C2
2050 DS(CC,I) = DS(CC,I - 1): N
EXT
2060  IF C2 > RX THEN RX = C2
2070  GOTO 2080
2080 MO = 1: GOSUB 70
2085  VTAB 23: HTAB 1: CALL  -
958  RETURN
2090  VTAB 23  HTAB 1  CALL
958: PRINT "ERRO NA DESTINACAO"
: FOR D = 1 TO 1000: NEXT
2100  GOTO 2080
2500  IF  PEEK (222) = 5 THEN
GOTO 1470
2510  END
3000  HOME : PRINT "TERMINO O P
ROGRAMA? (S/N) ";· GET AS
3010  IF AS <  > "S" THEN  GOSU
B 70· RETURN
3020  END
```



O que é uma planilha integrada?

Os usuários de microcomputadores com maior disponibilidade de memória RAM — entre eles os da linha PC e alguns modelos da linha Apple — podem ter o privilégio de utilizar os programas integrados de planilha eletrônica. Eles recebem esta denominação por incluírem, além do programa de planilha de cálculo propriamente dito, outros programas, como os de elaboração de gráficos (a partir dos dados da planilha), montagem e gestão de bancos de dados relacionais, e, em alguns tipos, processamento de texto.

A grande vantagem de sistemas desse tipo é a perfeita integração de dados em cada uma das aplicações. Além disso, eles tornam dispensável trocar programas, quando se deseja utilizar um outro aplicativo.

Para os micros da linha Apple que têm memória RAM de 128 Kbytes ou mais, já existe um pacote integrado: o *Totalworks*. Mas os pacotes mais difundidos são os disponíveis para os micros da linha PC (de dezesseis bits), tais como o *Lotus 1 2-3*, o *Framework* e o *Symphony*.

PADRÕES NATURAIS

- FAMÍLIAS DE CURVAS
- ENVOLTÓRIAS
- FOCOS E CÚSPIDES
- PADRÕES DE PONTOS
- DINÂMICA DE POPULAÇÕES

Com rotinas gráficas muito simples, podemos criar uma variedade surpreendente de desenhos — todos eles estruturados a partir de padrões fornecidos pela natureza.

A utilização do computador para desenhar a trajetória de objetos em queda livre sob a ação da gravidade foi examinada nos artigos publicados nas páginas 766 e 781. Com os programas apresentados, ilustramos alguns aspectos de um velho ramo da Física chamado *dinâmica* e vimos como traçar parábolas, círculos e elipses. Neste artigo, aprenderemos a desenhar outros tipos de curva usando técnicas semelhantes. Rotinas gráficas bastante simples se encarregarão dessa tarefa.

Nos últimos anos, a dinâmica recebeu um novo impulso enquanto campo de pesquisa. Um dos motivos para isso foi o reconhecimento de que seus princípios matemáticos não se aplicam somente ao movimento dos corpos sob a ação de determinadas forças. Cientistas ópticos, interessados na refração da luz das estrelas na atmosfera, químicos industriais que estudam o desencadeamento de reações e biólogos preocupados com o crescimento de populações animais que competem entre si — todos têm recorrido aos conhecimentos acumulados por esse ramo da Física. Além disso, os computadores — permitindo a repetição rápida de operações elementares — favoreceram a revelação de funções de uma complexidade estrutural que os antigos métodos de cálculo sequer sugeriam. Com freqüência, um padrão ou curva tem que ser repetido inúmeras vezes antes que possamos notar a presença de uma estrutura subjacente. Os programas abaixo demonstram isso.



FAMÍLIAS DE CURVAS

Algumas curvas, isoladamente simples, criam padrões bastante complexos quando são reunidas, constituindo uma família. Veja um exemplo em que se utilizam parábolas.

```
5 LET A$="": FOR N=1 TO 64:
LET A$=A$+" ": NEXT N
10 BRIGHT 1: BORDER 0: INK 5:
PAPER 0: CLS
20 LET NMAX=81
30 LET DELT=.05
40 LET SX=170/SQR 3: LET SY=
175
50 FOR N=1 TO NMAX
60 LET A=PI*(-1+2*N/NMAX)
70 PLOT 128,80
80 FOR T=0 TO 3 STEP DELT
90 LET X=T*COS A: LET Y=T*(
SIN A-T/2)
100 IF Y<=-.4 THEN GOTO 120
110 LET DX=SX*X+128: LET DY=SY
*Y+80
111 IF DX<0 OR DX>255 OR DY<0
OR DY>175 THEN GOTO 120
115 DRAW DX-PEEK 23677,DY-PEEK
23678
117 PRINT AT 19,0;A$
120 NEXT T
130 NEXT N
```



```
10 COLOR 15,1,1:SCREEN 2
20 NM=81:PI=4*ATN(1)
30 DE=.05
40 SX=160/SQR(3):SY=230
50 FOR N=1 TO NM
60 A=PI*(-1+2*N/NM)
70 DRAW"BM127,118"
80 FOR T=0 TO 3 STEP DE
90 X=T*COS(A):Y=T*(SIN(A)-T/2)
100 IF Y>-.4 THEN LINE-(SX*X+12
7,118-SY*Y) ELSE T=3
110 NEXT T
120 NEXT N
130 GOTO 130
```



```
10  HGR2
20 NM = 81:PI = 4 *  ATN (1)
30 DE = .05
40 SX = 160 /  SQR (3):SY = 180
50  FOR N = 1 TO NM
60 A = PI * ( - 1 + 2 * N / NM)
70  HCOLOR= 3:X1 = 139:Y1 = 118
80  FOR T = 0 TO 3 STEP DE
90 X = T *  COS (A):Y = T * ( S
IN (A) - T / 2)
100  IF Y >  - .35 THEN  HPLOT
X1,Y1 TO SX * X + 139,118 - SY
* Y:X1 = SX * X + 139:Y1 = 118
- SY * Y: GOTO 110
105 T = 3
110  NEXT T
120  NEXT N
```



```
10 PMODE 4,1:PCLS 1:SCREEN 1,0
20 NM=81:PI=4*ATN(1)
30 DE=.05
40 SX=160/SQR(3):SY=230
50 FOR N=1 TO NM
60 A=PI*(-1+2*N/NM)
70 DRAW"BM127,118"
80 FOR T=0 TO 3 STEP DE
90 X=T*COS(A):Y=T*(SIN(A)-T/2)
100 IF Y>-.4 THEN LINE-(SX*X+12
```

```
7,118-SY*Y),PRESET ELSE T=3
110 NEXT T
120 NEXT N
130 GOTO 130
```

O programa simula as trajetórias de jatos de água lançados por um esguicho de jardim, ou, em uma interpretação mais moderna, de nêutrons lançados de um tubo fino ligado a um reator. Em ambos os casos, a família de curvas é constituída por todas as trajetórias parabólicas que emergem no mesmo ponto, em direções variadas mas com a mesma velocidade. O padrão formado por essa família é a curva exterior que todas as curvas tocam. A curva exterior, chamada de *envoltória*, é também, neste caso, uma parábola. Em balística, usa-se o termo parábola limitante para uma curva desse tipo, pois ela define os limites da região que pode ser alcançada por projéteis que tenham uma determinada velocidade inicial.

É importante notar que a envoltória é uma propriedade da família de curvas como um todo, não tendo nenhum significado para uma trajetória em particular. Trata-se de um bom exemplo de que o todo pode ser mais do que a simples soma de suas partes.

No programa, a equação das curvas aparece na linha 90. **X** é a distância horizontal; **Y**, a distância vertical; **T** corresponde ao tempo (começando por zero no instante do lançamento) e cada trajetória tem uma direção determinada por **A**, o ângulo ou inclinação do lançamento. O programa desenha **NMAX** ou **NM** inclinações; tente mudar o valor dessa variável na linha 20.

FOCOS E CÚSPIDES

Este programa mostra como também as linhas retas podem constituir famílias com envoltórias interessantes.

```
10 BORDER 0: INK 7: PAPER 0:
CLS : LET N=2
30 LET Y0=80/N/N
40 PLOT 0,-6-Y0*2
50 FOR X=0 TO 255 STEP 2
60 LET Y=169-Y0-Y0*COS (N*2*
PI*X/255)
70 DRAW X-PEEK 23677,175-Y-
PEEK 23678
80 LET XT=X+2*Y0*N*PI*Y/255*
SIN (N*2*PI*X/255)
85 LET Y1=0: IF XT<0 THEN
LET XT=0: LET Y1=Y+255*X/(2*N
*PI*Y0*SIN (N*2*PI*X/255))
86 IF XT>255 THEN  LET XT=255
: LET Y1=Y-(255-X)*255/(2*N*
PI*Y0*SIN (N*2*PI*X/255))
90 DRAW XT-PEEK 23677,175-Y1-
PEEK 23678
100 PLOT X,175-Y
110 NEXT X
120 GOTO 120
```



```
10 COLOR 15,1,1:SCREEN 2
20 N=2:PI=4*ATN(1)
30 Y0=80/N/N
40 DRAW"BM0,"+STR$(INT(186-Y0*2
))
50 FOR X=0 TO 255 STEP 2
60 Y=186-Y0-Y0*COS(N*2*PI*X/255
)
70 LINE -(X,Y)
80 XT=X+2*Y0*N*PI*Y/255*SIN(N*2
*PI*X/255)
85 Y1=0:IF XT<0 THEN XT=0:Y1=Y+
255*X/(2*N*PI*Y0*SIN(N*2*PI*X/2
55))
86 IF XT>255 THEN XT=255:Y1=Y-(
255-X)*255/(2*N*PI*Y0*SIN(N*2*P
I*X/255))
90 LINE -(XT,Y1)
100 DRAW"BM"+STR$(X)+","+STR$(I
NT(Y))
110 NEXT
120 GOTO 120
```



```
10  HGR2
20 N = 2:PI = 4 *  ATN (1)
30 Y0 = 80 / N / N
40  HCOLOR= 3:XX = 0:YY = 186 -
 Y0 * 2
50  FOR X = 0 TO 279 STEP 2
60 Y = 186 - Y0 - Y0 *  COS (N
 * 2 * PI * X / 279)
70  HPLOT XX,YY TO X,Y
80 XT = X + 2 * Y0 * N * PI * Y
 / 279 *  SIN (N * 2 * PI * X /
 279)
85 Y1 = 0: IF XT < 0 THEN XT =
0:Y1 = Y + 279 * X / (2 * N * P
I * Y0 *  SIN (N * 2 * PI * X /
 279))
86  IF XT > 279 THEN XT = 279:Y
1 = Y - (279 - X) * 279 / (2 *
N * PI * Y0 *  SIN (N * 2 * PI
* X / 279))
90  HPLOT X,Y TO XT,Y1
100 XX = X:YY = Y
110  NEXT
```

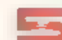


```
10 PMODE 4,1:PCLS1:SCREEN 1,0
20 N=2:PI=4*ATN(1)
30 Y0=80/N/N
40 DRAW"BM0,"+STR$(INT(186-Y0*2
))
50 FOR X=0 TO 255 STEP 2
60 Y=186-Y0-Y0*COS(N*2*PI*X/255
)
70 LINE -(X,Y),PSET
80 XT=X+2*Y0*N*PI*Y/255*SIN(N*2
*PI*X/255)
85 Y1=0:IF XT<0 THEN XT=0:YZ=Y+
255*X/(2*N*PI*Y0*SIN(N*2*PI*X/2
55))
86 IF XT>255 THEN XT=255:Y1=Y-(
255-X)*255/(2*N*PI*Y0*SIN(N*2*P
I*X/255))
90 LINE -(XT,YQ),PRESET
100 DRAW"BM"+STR$(X)+","+STR$(I
NT(Y))
110 NEXT
120 GOTO 120
```

O programa simula o reflexo de raios de luz sobre uma superfície ondulada — como o sol batendo nas águas de uma lagoa. Esta família é composta somente por linhas retas, que formam ângulos retos com uma curva senóide. A envoltória consiste na curva constituída pelos pontos focais. Estes, no caso dos raios, são mais intensos nos vales da senóide, onde a envoltória tem pontos brilhantes chamados *cúspides*. A existência de cúspides pode ser prevista através de um novo ramo da Matemática denominado *Teoria das Catástrofes* (esse nome dramático vem da aplicação da teoria ao desabamento de pontes e emborcação de navios).

No programa, a senóide é especificada na linha 60 e os raios de luz, na 80. O número de vales da senóide é **N** (menos no Apple). Tente mudar o valor de **N** na linha 20 (recomendamos N = 1)

Senóides também podem constituir uma família de curvas. Veja o programa apresentado a seguir.



```
10 BORDER 0: PAPER 0: INK 7:
CLS
20 LET QM=SQR (2*LN (3))
30 FOR Q=-QM TO QM*1.001 STEP
QM/30
40 PLOT 0,75+Q*75/QM
50 FOR T=0 TO 3*PI STEP .2
60 LET X=Q*COS (EXP (-Q*Q/2)*
T)
70 DRAW (T*240/3/PI)-PEEK
23677,75+(X*75/QM)-PEEK 23678
80 NEXT T
90 NEXT Q
```



```
10 COLOR 15,1,1:SCREEN 2
20 QM=SQR(2*LOG(3)):PI=4*ATN(1)
30 FOR Q=-QM TO QM*1.001 STEP Q
M/30
40 DRAW "BM0,"+STR$(INT(100-Q*9
0/QM))
50 FOR T=0 TO 3*PI STEP .2
60 X=Q*COS(EXP(-Q*Q/2)*T)
70 LINE -(T*240/3/PI,100-X*90/QM
)
80 NEXT T
90 NEXT Q
100 GOTO 100
```



```
10  HGR2
20 QM =  SQR (2 *  LOG (3)):PI
= 4 *  ATN (4)
```

```
30  FOR Q =  - QM TO QM * 1.001
 STEP QM / 30
40  HCOLOR= 3:X1 = 0:Y1 = 100 -
 Q * 90 / QM
50  FOR T = 0 TO 3.5 * PI STEP
 2
60 X = Q *  COS ( EXP (   Q * Q
 / 2) * T)
70  HPLOT X1,Y1 TO T * 240 / 3
 / PI,100 - X * 90 / QM:X1 = T *
 240 / 3 / PI:Y1 = 100 - X * 90
 / QM
80  NEXT T
90  NEXT Q
```



```
10 PMODE 4,1:PCLS 1:SCREEN 1,0
20 QM=SQR(2*LOG(3))·PI=4*ATN(1)
30 FOR Q= QM TO QM*1.001 STEP Q
M/30
40 DRAW"BM0,"+STR$(INT(100-Q*90
/QM))
50 FOR T=0 TO 3*PI STEP  2
60 X=Q*COS(EXP(-Q*Q/2)*T)
70 LINE  (T*240/3/PI,100-X*90/Q
M),PRESET
80 NEXT T
90 NEXT Q
100 GOTO 100
```

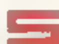
Simulamos aqui raios de luz atravessando uma fibra óptica cujo índice de refração varia ao longo de sua largura. Em uma escala microscópica, poderíamos estar representando também "trajetórias" de elétrons entre dois planos de átomos de um cristal colocado sob o canhão de um microscópio eletrônico. A família é composta por senóides que começam no canto esquerdo do vídeo, paralelas umas às outras; o período de cada curva depende de sua posição inicial. Embora esta família seja bem diferente da anterior, a envoltória também tem focos e cúspides.

No programa, as órbitas são especificadas na linha 60, que contém a distância **X** do eixo principal no tempo **T**. As senóides são definidas por **Q**. Como está, o programa desenha trinta senóides; para obter um número maior ou menor, modifique a linha 30.

PADRÕES DE PONTOS

Juntar curvas em famílias não é a única maneira de obter padrões interessantes. Um outro caminho é seguir uma órbita por um longo tempo, de modo que uma estrutura complexa possa se revelar, ainda que a fórmula matemática da curva seja relativamente simples. Para desenhar esse tipo de curva, não devemos fazer um traçado contínuo, pois, após alguns segundos, teremos preenchido a tela com um emaranhado de curvas. O mais conveniente é traçar a órbita por meio de pontos, dispostos na tela a intervalos regulares — como se a trajetória fosse observada sob luz estroboscópica. Os três programas seguintes exemplificam essa técnica. A posição dos pontos na tela nem sempre equivale à posição real do objeto de estudo; o que o programa fornece, muitas vezes, é uma representação abstrata, na qual a coordenada horizontal corresponde à posição real e a coordenada vertical, à velocidade.

Este programa leva vários minutos para completar o desenho:



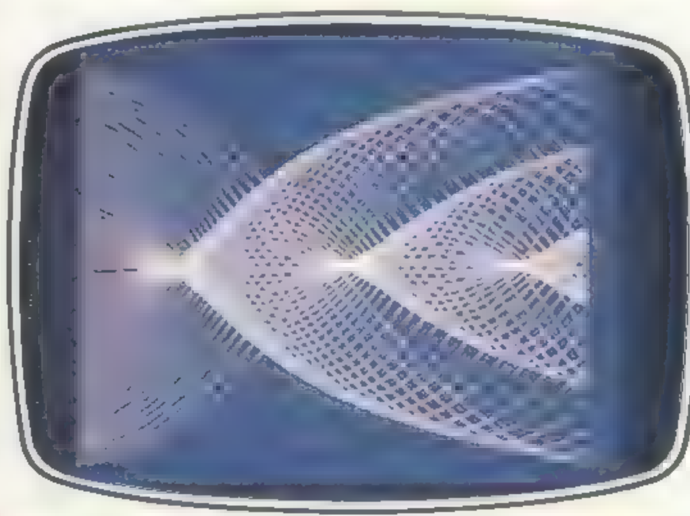
```
10 BORDER 0: PAPER 0: INK 7·
CLS
30 LET A=76.11
40 LET ALF=A*PI/180: LET C=
COS (ALF)
50 LET S=SIN (ALF)
60 LET NMAX=200
70 LET M=52
80 FOR J=1 TO M
90 LET X=0· LET Y=J/M
100 FOR N=1 TO NMAX
110 LET W=X
120 LET X=X*C (Y-X*X)*S  LET Y
=W*S+(Y W*W)*C
130 IF ABS (X)>4 OR ABS (Y)>4
THEN  GOTO 860
135 IF X>1 OR Y>1 THEN  GOTO
 150
140 PLOT X*128+128,Y*85+85
150 NEXT N
160 NEXT J
```



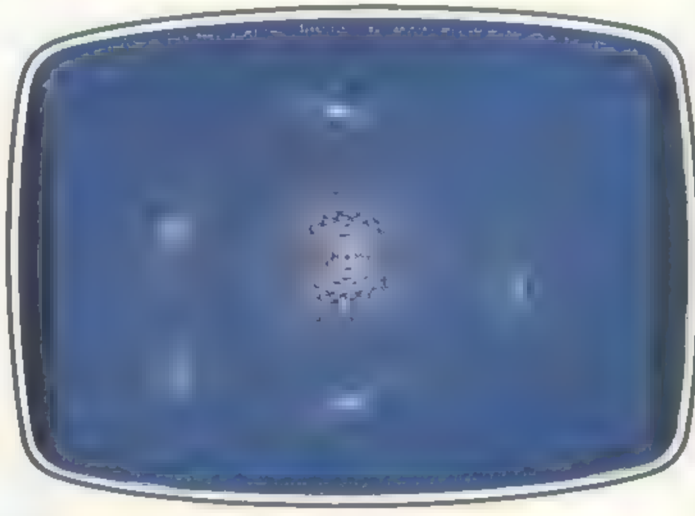
```
10 COLOR 15,1,1·SCREEN 2
20 A·76 11 PI=4*ATN(1)
30 AL=A*PI/180·C=COS(AL)
40 S SIN(AL)
50 NM=200
60 M=52
70 FOR J=1 TO M
80 X=0·Y=J/M
90 FOR N=1 TO NM
100 W=X
110 X=X*C-(Y-X*X)*S Y=W*S+(Y W*
W)*C
120 IF ABS(X)>4 OR ABS(Y)>4 THE
N 160
125 IF ABS(Y)>1 OR ABS(X)>1 THE
N 140
130 PSET(128+X*128,96-Y*96)
140 NEXT N
150 NEXT J
160 GOTO 160
```



```
10  HGR2
20 A = 76.11·PI = 4 *  ATN (1)
30 AL = A * PI / 180 C =  COS (
AL)
40 S =  SIN (AL)
50 NM = 200
60 M = 52
70  FOR J = 1 TO M
```



Raios de luz em uma fibra óptica.



"Ilhas de pontos" ou linhas de força

```
80 X = 0:Y = J / M
90  FOR N = 1 TO NM
100 W = X
110 X = X * C - (Y - X * X) * S
:Y = W * S + (Y - W * W) * C
120  IF  ABS (X) > 4 OR  ABS (Y
) > 4 THEN 160
125  IF  ABS (X) > 1 OR  ABS (Y
) > 1 THEN 140
130  HPLOT 128 + X * 128,96 - Y
 * 96
140  NEXT N
150  NEXT J
160  GOTO 160
```

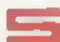


```
10 PMODE 4,1 PCLS1 SCREEN 1,0
20 A=76 11 PI=4*ATN(1,
30 AL=A*PI/180 C=COS(AL)
40 S=SIN(AL)
50 NM 200
60 M 52
70 FOR J 1 TO M
80 X 0 Y=J/M
90 FOR N=1 TO NM
100 W=X
110 X=X*C (Y X*X)*S Y W*S+(Y-W*
W)*C
120 IF ABS(X)>4 OR ABS(Y)>4 THE
N 160
125 IF ABS(Y)>1 OR ABS(X)>1 THE
N 140
130 PRESET(128+X*128,96 Y*96)
140 NEXT N
150 NEXT J
160 GOTO 160
```

O programa simula o movimento de partículas subatômicas (como prótons e elétrons) em um acelerador ou, ainda, linhas de força de um campo magnético. Sua atuação, em termos matemáticos, consiste em tomar uma série de pontos iniciais e movê-los pela tela, seguindo esta regra: "faça uma rotação em torno do centro da tela, segundo um ângulo fixo, só que com uma ligeira modificação". Sem esta "ligeira modificação", todas as curvas seriam círculos concêntricos — e, de fato, as mais próximas do centro se assemelham a círculos. O efeito dessa modificação tem maior impacto sobre as curvas externas, que acabam formando um conjunto de "ilhas". Se pudéssemos ampliar mais o desenho, distinguiríamos várias ilhas menores, distribuídas entre as que podemos observar agora.

No programa, o número de pontos iniciais é **M**, na linha 60: caso você não queira esperar tanto para ver o desenho completo, diminua esse valor. Cada ponto é desenhado **NMAX** ou **NM** vezes; o valor é especificado na linha 50. A regra fica nas linhas 100 e 110, que determinam de que modo as coordenadas vertical e horizontal serão modificadas a cada repetição. O ângulo fixo de rotação é dado por **A** em graus, na linha 20; experimente valores diferentes (recomendamos 90).

O programa apresentado a seguir gera um padrão surpreendente, a partir de um ponto inicial:



```
20 LET K=51 3: BORDER 0: INK
7: PAPER 0: CLS
30 LET X=1/PI: LET P=0
40 LET A=1/SQR 5
50 FOR N=1 TO 10000
60 LET Y=X-.5: LET X=X+A-INT
(X+A)
70 LET P=P-Y
80 PLOT X*255,P*K+85
90 NEXT N
```



```
10 COLOR 15,1,1:SCREEN 2
20 K=60 X=1/(4*ATN(1)):P=0
30 A=1/SQR(5)
40 FOR N=1 TO 10000
50 Y=X-.5:X=X+A-INT(X+A)
60 P=P-Y
70 PSET(X*255,128-P*K)
80 NEXT N
90 GOTO 90
```



```
10  HGR2 :K = 40
20 X = 1 / (4 *  ATN (1)):P = 0
30 A = 1 /  SQR (5)
40  FOR N = 1 TO 10000
50 Y = X - .5:X = X + A -  INT
(X + A)
60 P = P - Y
70  HCOLOR= 3: HPLOT X * 279,10
0 - P * K
80  NEXT N
```



```
10 PMODE 4,1 PCLS1 SCREEN 1,0 K
=60
20 X=1/(4*ATN(1)) P=0
30 A=1/SQR(5)
40 FOR N=1 TO 10000
50 Y=X-.5 X=X+A-INT(X+A)
60 P=P-Y
70 PRESET (X*255,128-P*K)
80 NEXT N
90 GOTO 90
```

Esse programa é uma simulação da *ressonância*, que ocorre quando dois fenômenos físicos têm frequências coincidentes ou múltiplas. Por exemplo, um dos fenômenos pode ser a órbita de um asteróide em torno do sol; o segundo, a perturbação dessa órbita pela atração gravitacional de Júpiter. Podemos nos perguntar: as perturbações causadas pelo grande planeta serão capazes de arrancar o asteróide de sua órbita? Tudo depende da *razão* entre as duas frequências (a razão é obtida dividindo-se uma pela outra). As partículas dos anéis de



Uma regra simples produz o caos.



Mudanças em uma população de peixes.

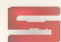
Saturno estão sujeitas a esse tipo de efeito.

No programa, a razão é chamada de **A** e seu valor (menor que 1) está na linha 30. O número de repetições da ressonância é 10000 (final da linha 40); reduza esse valor caso não queira esperar muito. A transformação da ressonância, nas linhas 50 e 60, é obtida por meio de uma regra que modifica as posições horizontal (**X**) e vertical (**P**) dos pontos na tela.

Um delicado padrão em "cortina" é obtido com **A = 1/SQR(5)**, ou seja, 0.4472136. Como esse valor não corresponde à razão entre dois números inteiros (em termos do computador, trata-se de um número irracional), ele é não-ressonante. Para chegar à ressonância, temos que atribuir a **A** um valor que seja a razão entre dois números inteiros — como 9/20, que vale 0.45. Tente também um outro número irracional, como **1/PI**. Se quiser reduzir a escala vertical, diminua o valor de **K**.

CAOS

Observe que certas regras simples não produzem nenhum padrão. O próximo programa ilustra o fenômeno.



```
10 BORDER 0: PAPER 0: INK 7:
CLS
20 LET X=1/PI
30 LET Y=1/PI
40 FOR M=1 TO 10000
50 LET X=X+Y-INT (X+Y)
60 LET Y=X+Y-INT (X+Y)
70 PLOT X*255,Y*175
80 NEXT M
```



```
10 COLOR 15,1,1:SCREEN 2
20 PI=4*ATN(1):X=1/PI
30 Y=1/PI
40 FOR M=1 TO 10000
50 X=X+Y-INT(X+Y)
60 Y=X+Y-INT(X+Y)
70 PSET(X*255,192-Y*191)
80 NEXT M
90 GOTO 90
```



```
10  HGR2
20 PI = 4 * ATN (1):X = 1 / PI
30 Y = 1 / PI
40  FOR M = 1 TO 10000
50 X = X + Y -  INT (X + Y)
60 Y = X + Y -  INT (X + Y)
70  HCOLOR= 3: HPLOT X * 279,19
2 - Y * 191
80  NEXT M
```



```
10 PMODE 4,1:PCLS1:SCREEN 1,0
20 PI=4*ATN(1):X=1/PI
30 Y=1/PI
40 FOR M=1 TO 10000
50 X=X+Y-INT(X+Y)
60 Y=X+Y-INT(X+Y)
70 PRESET(X*255,192-Y*191)
80 NEXT M
90 GOTO 90
```

Esse programa simula o vaivém errático das bolas de um fliperama, o movimento das moléculas de um gás, ou qualquer outro sistema de órbita tão imprevisível que se torna impossível distingui-la de um processo aleatório. Convém notar, porém, que os 10.000 pontos desenhados na tela não são aleatórios, resultando da repetição de uma regra determinística, ou seja, que não abriga nenhum elemento de acaso. A regra considera a tela como um quadrado unitário onde as coordenadas verticais e horizontais variam entre zero e um. Ela opera em três etapas, nas linhas 50 e 60 do programa. Primeiro, as coordenadas **X** e **Y** do último ponto são somadas para produzir um novo **X**; depois, esse **X** é somado a **Y**, resultando em outro **Y**; finalmente, se **X** ou **Y** forem maiores que 1, um número inteiro apropriado é subtraído para que as coordenadas permaneçam no intervalo válido.

POPULAÇÕES DE PEIXES

Como a população de peixes de um lago varia ao longo das gerações? Isto depende dos fatores que condicionam as mudanças populacionais de uma geração para a outra. Para definir uma regra, devemos considerar tanto a tendência ao crescimento populacional resultante da reprodução, como a tendência à redução, determinada pela quantidade de alimento disponível no lago. Conforme o equilíbrio estabelecido entre esses dois fatores, a população pode ficar estável, alternar entre um ou mais valores, ou variar de tamanho ao acaso, ao longo das gerações.

O programa dado a seguir ilustra o fenômeno. A posição horizontal, **A**, corresponde à razão entre disponibilidade de alimento e taxa de natalidade, representando assim as sucessivas gerações. A posição vertical **Y** corresponde ao tamanho da população, que aumenta ou diminui alguns pontos conforme as gerações se sucedem. O tamanho da população **Y** só é colocado na tela quando se encontra em equilíbrio, isto é, quando assume um valor estável ou alterna entre dois ou mais valores, que são denominados *pontos de atração*. No Spectrum e no TRS-Color, um som de freqüência proporcional ao tamanho da população é emitido.



```
10 BORDER 0: INK 7: PAPER 0:
CLS
20 LET S=.03: LET NMIN=50:
LET NMAX=80
30 FOR A=2.8 TO 4 STEP S
40 LET Y=1/PI
50 FOR N=1 TO NMAX
60 LET Y=A*Y*(1-Y)
70 IF N>NMIN THEN  PLOT 255*(
A-2.8)/1.2,Y*175
80 SOUND .0075,Y*20
90 NEXT N
100 NEXT A
```



```
10 COLOR 15,1,1:SCREEN 2
20 S=1/127:NN=50:NX=80
30 FOR A=2.8 TO 4 STEP S
40 Y=.25/ATN(1)
50 FOR N=1 TO NX
60 Y=A*Y*(1-Y)
70 IF N>NN THEN PSET(255*(A-2.8
)/1.2,192-Y*191)
90 NEXT N
100 NEXT A
110 GOTO 110
```

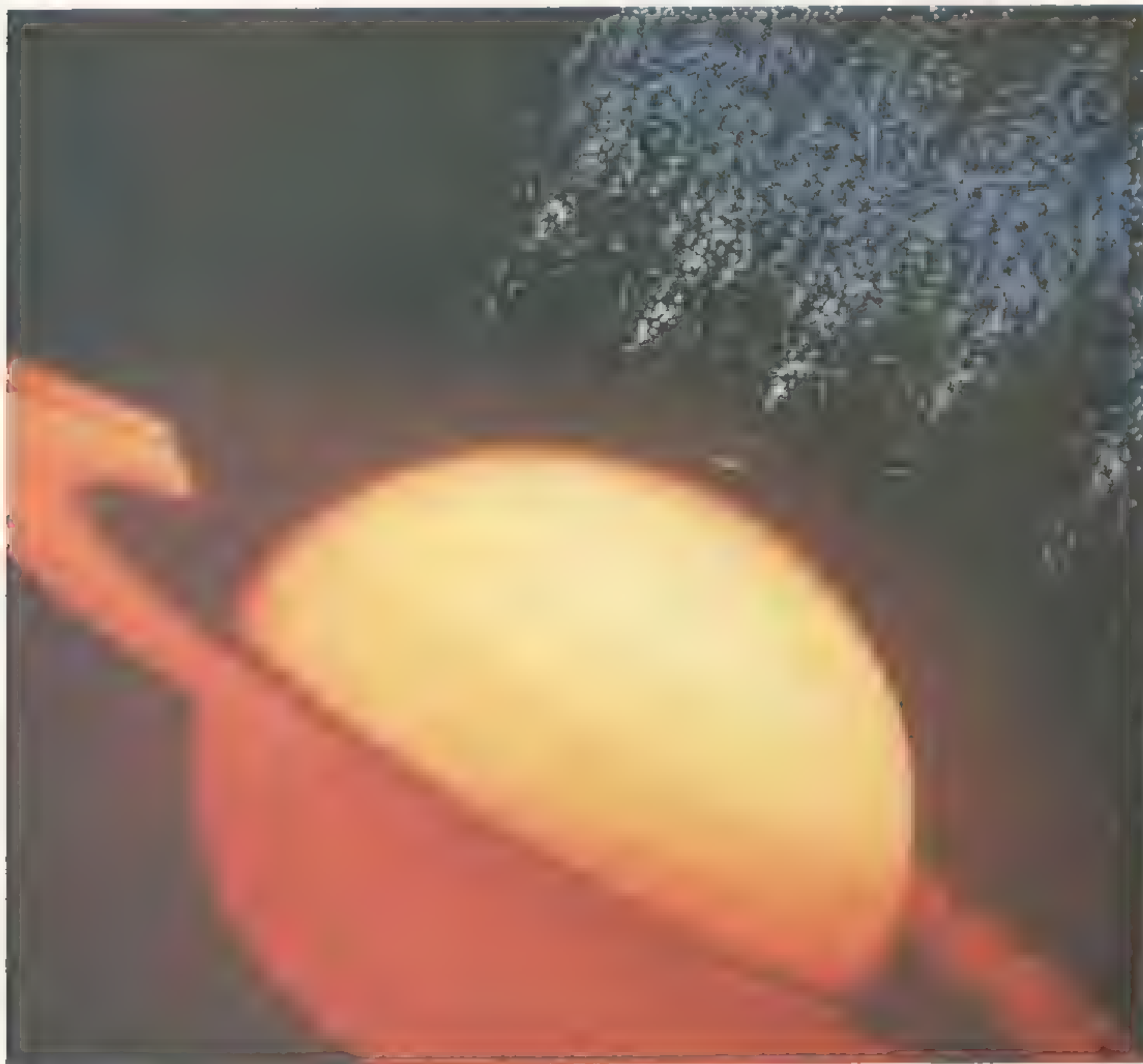


```
10  HGR2
20 S = 1 / 127:NN = 50:NX = 80
30  FOR A = 2.8 TO 4 STEP S
40 Y = .25 /  ATN (1)
50  FOR N = 1 TO NX
60 Y = A * Y * (1 - Y)
70  IF N > NN THEN  HPLOT 279 *
 (A - 2.8) / 1.2,192 - Y * 192
90  NEXT N
100  NEXT A
```



```
10 PMODE 4,1:PCLS1:SCREEN 1,0
20 S=1/127:NN=50:NX=80
30 FOR A=2.8 TO 4 STEP S
40 Y=.25/ATN(1)
50 FOR N=1 TO NX
60 Y=A*Y*(1-Y)
70 IF N>NN THEN PRESET (255*(A-
2.8)/1.2,192-Y*191)
80 SOUND 1+255*Y,1
90 NEXT N
100 NEXT A
110 GOTO 110
```

Como você pode observar, os pontos de atração modificam-se drasticamente à medida que a disponibilidade de alimento aumenta. No início (lado esquerdo da tela), só há um valor de atração, indicando uma população estável. Isso significa que o alimento é suficiente para os peixes sobreviverem e se reproduzirem. Se a população crescer demais,



haverá menos alimento e alguns peixes morrerão. Com a maior disponibilidade de alimento para os sobreviventes, a população volta a crescer, podendo, em seguida, passar por uma nova fase de estabilidade.

Subitamente, quando **A** assume um certo valor, a quantidade de alimento ultrapassa determinado limite. Passam a existir, então, dois pontos de atração e a curva se bifurca — indicando que o tamanho da população agora se alterna entre dois valores. Mais tarde, quando a quantidade de alimento se torna ainda maior, a curva se bifurca mais uma vez, aumentando o número de pontos de atração. Mais e mais divisões vão ocorrendo em uma seqüência infinita, da qual o programa só tem resolução para mostrar os primeiros passos. As bifurcações se acumulam e infinitos pontos de atração passam a corresponder a um certo valor finito de **A**. Em conseqüência, a população varia sem nunca se estabilizar.

A descoberta dessa "árvore de bifurcações", que acaba num completo caos, despertou grande interesse na comunidade científica, por ter diversas aplicações. Ela retrata, por exemplo, a modificação do fluxo de um líquido, que passa de um estágio moderado (ponto de atração estável) para um turbulento (caos), conforme a velocidade aumenta. O mesmo efeito pode ser observado na fumaça de um cigarro, que sobe suavemente e de repente desenvolve espirais, terminando em turbulência.

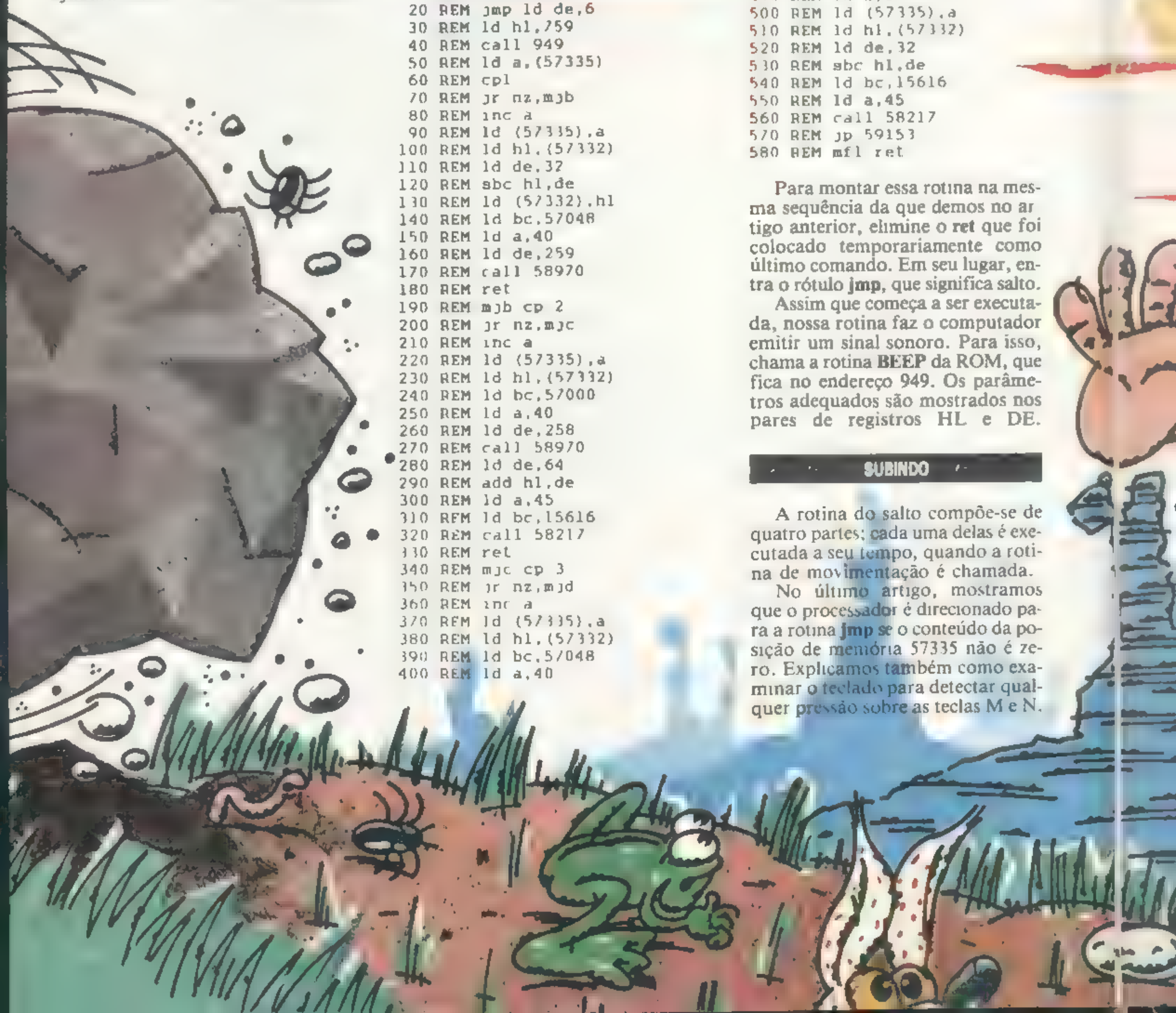
No programa, a lei de evolução está na linha 60. Para ampliar a resolução, altere a linha 20 de forma que S = .005, **NMIM** ou **NM** = 200 e **NMAX** ou **NX** = 300. No Spectrum e no TRS-Color, apague a linha 80 para aumentar a velocidade do programa.

AVALANCHE: OS SALTOS DE WILLIE

Nosso personagem não pode mesmo confiar em sua sorte. Surpreendido por uma avalanche tão logo começou a caminhada, conta apenas com um recurso para escapar das pedras: saltar sobre elas.

A parte do programa aqui apresentada permite que Willie se movimente para cima e para baixo, no mesmo lugar — ou seja, nosso pobre personagem aprende, a tempo, a saltar sobre as pedras que se aproximam.

Willie já pode andar, mas isso não o ajudará muito na hora de enfrentar a avalanche. Vamos fazê-lo saltar, de modo que as pedras passem rolando sob suas pernas, sem atingi-lo.



```
 10 REM org 59336
 20 REM jmp ld de,6
 30 REM ld hl,759
 40 REM call 949
 50 REM ld a,(57335)
 60 REM cpl
 70 REM jr nz,mjb
 80 REM inc a
 90 REM ld (57335),a
100 REM ld hl,(57332)
110 REM ld de,32
120 REM sbc hl,de
130 REM ld (57332),hl
140 REM ld bc,57048
150 REM ld a,40
160 REM ld de,259
170 REM call 58970
180 REM ret
190 REM mjb cp 2
200 REM jr nz,mjc
210 REM inc a
220 REM ld (57335),a
230 REM ld hl,(57332)
240 REM ld bc,57000
250 REM ld a,40
260 REM ld de,258
270 REM call 58970
280 REM ld de,64
290 REM add hl,de
300 REM ld a,45
310 REM ld bc,15616
320 REM call 58217
330 REM ret
340 REM mjc cp 3
350 REM jr nz,mjd
360 REM inc a
370 REM ld (57335),a
380 REM ld hl,(57332)
390 REM ld bc,57048
400 REM ld a,40
410 REM ld de,259
420 REM call 58970
430 REM ld de,32
440 REM add hl,de
450 REM ld (57332),hl
460 REM ret
470 REM mjd cp 4
480 REM jr nz,mfl
490 REM ld a,0
500 REM ld (57335),a
510 REM ld hl,(57332)
520 REM ld de,32
530 REM sbc hl,de
540 REM ld bc,15616
550 REM ld a,45
560 REM call 58217
570 REM jp 59153
580 REM mfl ret
```

Para montar essa rotina na mesma sequência da que demos no artigo anterior, elimine o **ret** que foi colocado temporariamente como último comando. Em seu lugar, entra o rótulo **jmp**, que significa salto.

Assim que começa a ser executada, nossa rotina faz o computador emitir um sinal sonoro. Para isso, chama a rotina **BEEP** da ROM, que fica no endereço 949. Os parâmetros adequados são mostrados nos pares de registros HL e DE.

SUBINDO

A rotina do salto compõe-se de quatro partes; cada uma delas é executada a seu tempo, quando a rotina de movimentação é chamada.

No último artigo, mostramos que o processador é direcionado para a rotina **jmp** se o conteúdo da posição de memória 57335 não é zero. Explicamos também como examinar o teclado para detectar qualquer pressão sobre as teclas M e N.

- AS QUATRO PARTES DA ROTINA DE SALTO
- SALTO VERTICAL E SALTO DIAGONAL
- EFEITOS SONOROS
- AJUSTAMENTO DOS APONTADORES DE TELA
- WILLIE ESTÁ NO AR
- DE VOLTA AO SOLO
- FIM DO SALTO



Se apenas a tecla N foi pressionada e Willie deve saltar, a posição de memória 57335 é carregada com 1, que irá direcionar o processador para a rotina aqui apresentada na próxima vez que o programa principal — que controla o jogo — chamar a rotina de movimentação. Caso as teclas M e N tenham sido pressionadas, Willie deve saltar para a frente (em diagonal); 129 é, então, carregado em 57335. Esta situação será devidamente analisada no próximo artigo da série *Avalanche*.

Por hora, imagine apenas que a rotina de movimentação de Willie constatou que houve uma pressão sobre a tecla N, na última vez que foi chamada. Agora ela voltou a ser chamada, o que fez o processador direcionar-se exatamente para nossa rotina. Depois da emissão do som que caracteriza o início de sua execução...

O conteúdo da posição de memória 57335 é carregado no acumulador outra vez. Embora ele já estivesse ali quando esta rotina foi chamada, é provável que a rotina **BEEP** tenha utilizado o acumulador para realizar o efeito sonoro do salto. Assim, convém carregar o valor em A novamente.

Depois, o conteúdo de A é comparado com 1. Naturalmente, esse valor está presente; em conseqüência, a próxima instrução, **jr nz,mjb**, não é executada e o processador continua com **inc a**. Isto incrementa o conteúdo do acumulador para 2, valor que é colocado de volta em 57335 por **ld (57335),a**. Incrementa-se, assim, o contador de salto, para que a rotina de movimentação acione a rotina **jmp** na próxima vez que for chamada. Nesse ponto, porém, a primeira parte do salto será deixada de lado; o processador executa a segunda parte. Como você deve ter reparado, cabe à instrução **cp 1**, seguida da instrução **jr nz,mjb**, fazer o processador saltar para a segunda parte.

A posição de Willie na tela é então carregada dos endereços 57332 e 57333 para o par de registros HL. O valor 32 é subtraído dessa posição e colocado em DE. Com a execução de **sbc hl, de**, o apontador se move para a posição imediatamente acima de Willie na tela. O resul-

tado é carregado de volta nos endereços 57332 e 57333, que contêm a posição do personagem na tela.

Embora essa posição tenha sido colocada nos endereços, ela ainda permanece no par HL. Todos os **ld** são essencialmente operações de cópia que colocam o valor na nova posição ou registro, mantendo esse valor na posição ou registro de origem. No nosso caso, isso é importante, pois, para chamar a rotina de impressão de bloco em 58970, a posição de impressão na tela precisa estar no par de registros HL.

O par BC, com o apontador de dados para a primeira figura de Willie saltando, é carregado em 57048. Esses dados foram apresentados em um artigo anterior de *Avalanche* e podem ser lidos numa tabela de dados na memória.

A é carregado com 40 — código de azul sobre ciano, a cor de Willie — e DE, com 259. Isso faz com que a rotina de impressão de bloco imprima um bloco um por três — 259 carrega 1 no registro D e 3 no registro E; 1 x 256 + 3 = 259. A rotina de impressão de bloco em 58970 é então chamada e imprime a figura de Willie saltando.

Em seguida, o processador retorna da rotina.

WILLIE ESTÁ NO AR

A próxima parte da rotina de salto é chamada quando se aciona novamente a rotina de movimentação. Mais uma vez, essa seção começa verificando se a posição de memória 57335 contém 2 —



o que acontecerá se ela estiver sendo chamada pela segunda vez.

Lembre-se de que o conteúdo de 57335 está no acumulador quando o processador salta para essa parte da rotina; portanto, você não precisa carregá-lo de novo. Se o valor 2 está presente, o processador continua com a rotina e torna a incrementar o contador de salto com as instruções **inc a** e **ld (57335),a.** Assim, quando a rotina for chamada na próxima vez, a instrução **cp 2** não irá encontrar um valor 2 e a instrução **jr nz,mjc** mandará o processador para a terceira parte.

Se o 2 estiver presente e esta parte da rotina for acionada, a nova posição de Willie no ar é carregada dos endereços 57332 e 57333 para o par de registros HL. BC é carregado com 57000, que corresponde ao endereço inicial dos dados para a figura de Willie de pé com as pernas juntas. Esta é a figura que será impressa quando, ao saltar, Willie estiver no alto.

A é carregado com 40 — azul sobre ciano, a cor de Willie. DE é ajustado com 258 para imprimir um bloco de um por dois — 1 x 256 + 2 = 258. A rotina de impressão em 58970 imprime Willie no ar com as pernas juntas.

O par DE é carregado com 64 e adicionado ao apontador de tela em HL — o que move esse apontador duas linhas para baixo. O par HL aponta agora para a cabeça de Willie. Lembre-se de que, subtraindo 64 (32 para cada linha), fazemos com que ele aponte para a posição abaixo dos pés do personagem. A é carregado com 45 — ciano sobre ciano, apenas um pedaço comum de céu. O apontador de dados BC é carregado com 15616, o início do conjunto de caracteres para um espaço em branco.

A rotina **print**, no endereço 58217, é chamada, imprimindo um espaço em branco para apagar a parte da figura de Willie que restou da primeira seção da rotina de salto. Lembre-se de que um bloco de um por três foi impresso ali, e, agora, apenas um bloco de um por dois. O processador retorna de novo, até a rotina voltar a ser chamada.

DESCENDO

Na próxima vez, o processador acionará a terceira parte da rotina de movimentação. Antes de mais nada, ele verifica se é necessário acionar a quarta parte da rotina de salto. Em seguida, o contador de salto, que ainda está no acumulador, é incrementado e armazenado novamente em 57335, para que o processador pule para a quarta parte da rotina na próxima vez.

O apontador da posição de Willie na tela é carregado de volta em HL. O apontador de dados em BC é ajustado com o mesmo valor utilizado na primeira parte da rotina de salto. Willie descerá ao chão pelo mesmo processo que subiu.

A é carregado de novo com 40 — azul sobre ciano; DE volta a ser ajustado com 259 — um por três. A rotina de impressão de bloco em 58970 é chamada e imprime Willie descendo.

O par DE é carregado com 32, valor adicionado ao par HL. O resultado é carregado de volta em 57332 e 57333. Isso move o apontador de tela uma posição para baixo.

O processador retorna agora com as posições dos dados ajustados para a próxima parte.

FIM DO SALTO

Mais uma vez a rotina começa com um **cp**, para verificar se esta é a parte necessária da rotina de salto. Mas, agora, estamos diante da última seção da rotina que faz Willie pular. Para que o processador chegue até aqui, o conteúdo do acumulador deve ser 4 (parte que finaliza o salto) ou 129, valor que indica que Willie irá saltar para a frente. Nesse caso, **jr nz,mfj** manda o processador para outra instrução de retorno, no final deste programa. O retorno será apagado mais tarde, servindo para indicar o local onde se inicia a rotina de salto à frente, que será fornecida no próximo artigo de *Avalanche*. Desta vez, entretanto, nenhum teste será feito e não precisaremos incrementar o contador de salto. É necessário que se ajuste o contador com 0, para que, quando a rotina de movimentação for novamente chamada, o processador execute a parte dada no artigo anterior e verifique se alguma tecla foi pressionada. Isso é feito por **ld a,0** e **ld (57335),a**.

A posição de Willie volta a ser carregada de 57332 e 57333 para o par de registros HL, onde 32 é subtraído através do par DE, fazendo o apontador se mover uma posição acima na tela.

BC é carregado com o endereço de um espaço vazio e A, com o código de ciano sobre ciano. A rotina **print**, em 58217, é chamada de novo. Apaga-se, assim, o caractere adicional ocupado por Willie, impresso na terceira parte da rotina de salto.

A instrução **jp 59153** faz o processador voltar para o início da rotina de movimentação, dada no artigo anterior. Como o contador de salto contém agora o valor 0, o processador executa a rotina que imprime Willie parado, apagando o que restou da antiga figura.

A última instrução **ret** da listagem tem a função de prevenir que ocorra um erro no programa, caso a rotina de salto para a frente venha a ser chamada. Essa instrução será apagada na próxima parte do programa.



```
10   ORG 20140
20   JUM JSR CLICK
30   LDA 18261
40   CMPA #1
50   BNE MJA
60   INC 18261
70   LDX 18249
80   PSHS X
90   LEAX 256,X
100   LDU #1536
110   JSR CHARPR
120   PULS X
130   LEAX -256,X
140   STX 18249
150   LDU #17814
160   JSR CHARPR
170   LEAX 254,X
180   LDU #17846
190   JSR CHARPR
200   RTS
210   MJA CMPA #2
220   BNE MJB
230   INC 18261
240   LDX 18249
250   LEAX  256,X
260   LDU #17870
270   JSR CHARPR
280   LEAX 254,X
290   JSR CHARPR
300   LEAX 254,X
310   JSR CHARPR
320   RTS
330   MJB CMPA #3
340   BNE MJC
350   INC 18261
360   LDX 18249
370   LEAX -256,X
380   LDU #1536
390   JSR CHARPR
400   LEAX 254,X
410   LDU #17926
```

```
420  JSR CHARPR
430  LEAX 254,X
440  LDU #17958
450  JSR CHARPR
460  LEAX 254,X
470  LDU #17990
480  JSR CHARPR
490  RTS
500  MJC CMPA #4
510  BNE MFJ
520  CLR 18261
530  LDX 18249
540  PSHS X
550  LDU #1536
560  JSR CHARPR
570  PULS X
580  LEAX 256,X
590  STX 18249
600  LDU #17774
610  JSR CHARPR
620  LEAX 254,X
630  JSR CHARPR
640  RTS
650  MFJ RTS
660  CHARPR EQU 19402
670  CLICK EQU 20847
```

A primeira coisa que a rotina **JUM** (ou de salto) faz é chamar a sub-rotina **CLICK**, que toca uma das notas do efeito sonoro do salto. Se você ainda não tiver digitado essa rotina, adicione um retorno em 20847 por meio da instrução **POKE**, com 57 nesse endereço. Assim, não haverá erro quando o programa for testado.

PULANDO POR ETAPAS... JÁ!

O acumulador é carregado com o conteúdo da posição de memória 18261, que contém a variável do salto. Inicialmente, ela está carregada com 1 ou 129 — dependendo de qual tecla foi pressionada — indicando, portanto, se Willie deve pular verticalmente ou dar um salto à frente. Cada um desses tipos de pulo é dividido em quatro etapas, executadas uma por vez, quando a rotina de movimentação é chamada. Para garantir o acionamento da etapa correta, a variável do salto é incrementada na execução de cada parte.

Se a variável do salto é 0, utiliza-se a seção da rotina que examina o teclado para ver se uma ordem de pular foi dada. Caso a variável de movimentação contenha qualquer outro valor que não seja 0, o processador vem para esta parte do programa e executa uma das rotinas de salto.

Se for 1, a primeira parte da rotina é acionada. Durante sua execução, a variável de salto é incrementada, para que, na próxima vez que a rotina de movimentação for chamada, essa seção seja evitada e o processador passe para a parte dois. Assim, sempre que a rotina de movimentação for acionada, ela chamará a próxima parte — até que a última (a quarta) seja executada e a variável do salto volte a ser ajustada com 0.

Se a última rotina ajustou a variável do salto com 129, esta parte é ignorada. A rotina do salto para a frente, que daremos no próximo artigo de *Avalanche*, é então acessada.

UM!

O conteúdo do acumulador é comparado com 1. Se 1 não está presente, o processador passa para **MJA**. Caso contrário, o processador continua com esta parte da rotina.

A primeira coisa que ela faz é incrementar a variável em 18261, deixando-a pronta para a próxima vez.

X é carregado com a posição de Willie, que está no endereço 18249 e é armazenado temporariamente na pilha. O apontador em X é incrementado com 256, movendo-se um caractere para baixo. U é carregado com 1536, o endereço do canto superior esquerdo da tela. O processador chama então a rotina **CHARPR**, que imprime um bloco de céu azul sobre a metade inferior de Willie. Se não fizermos isso, as pernas de Willie aparecerão em sua posição anterior.

A posição de Willie é novamente guardada na pilha e movida um caractere para cima pela subtração de 256. O resultado é armazenado de volta em 18249. U é carregado com 17814, endereço inicial para os dados de Willie com as pernas abertas. Em seguida, a rotina **CHARPR** é chamada para imprimir a metade superior.

O apontador de tela é incrementado com 254, passando a apontar para o início da metade inferior de Willie. U é recarregado e a rotina **CHARPR** volta a ser chamada para imprimir sua metade inferior.

O processador retorna depois de ter imprimido Willie com as pernas abertas um caractere acima do chão.

DOIS!

Quando a rotina de movimentação de Willie for novamente chamada, o número no endereço 18261 será 2. Portanto, o processador irá pular a primeira parte da rotina publicada no artigo anterior e a parte desta rotina que foi dada até agora. Mas, quando ele saltar de novo para **MJA**, o conteúdo de 18261, que ainda está no acumulador, será comparado com 2. Na chamada seguinte da rotina de movimentação, quando o conteúdo do endereço 18261 tiver sido incrementado mais uma vez, a instrução **BNE MJB** mandará o processador verificar se seu valor é 3, 4, 129, 130, 131 ou 132. Desta vez ele executará a próxima seção da rotina, imprimindo a figura de Willie no segundo estágio do pulo.

Essa seção também começa incrementando o conteúdo de 18261, acertando-o para a próxima chamada da rotina de salto. Depois, a nova posição de Willie é carregada de 18249 para X e decrementada de 256 — o que faz o personagem se mover um caractere acima na tela. Mas agora o resultado não é colocado de volta em 18261. Willie está no alto de seu pulo.

U é carregado com 17870, endereço inicial dos dados da nova figura de Willie. Desta vez, ele tem três caracteres de altura, impressos em três posições — **CHARPR** precisa ser chamada três vezes e X é decrementado com 254 para mover o apontador uma linha abaixo em cada chamada.

Feito isso, o processador retorna.

TRÊS!

Quando a rotina de movimentação volta a ser chamada, o conteúdo de 18261 é 3 e o processador executa a rotina **MJB**. Ela começa, mais uma vez, incrementando 18261, carregando 18249 em X e subtraindo 256.

Só que agora a figura anterior de Willie deve ser apagada. U é carregado com 1536, o endereço do canto superior esquerdo da tela, e a rotina **CHARPR** é chamada, imprimindo a cabeça de Willie na tela. O apontador X é incrementado para indicar a posição inferior na tela e U é carregado com o endereço dos dados para a impressão do resto da figura.

QUATRO!

Se nosso personagem está pulando na vertical, mais cedo ou mais tarde o processador irá executar **MJC**. As instruções **CMPA #4** e **BNE** chamarão **MJF** apenas quando Willie for saltar na diagonal. Mas aquela pequena rotina será sempre executada.

Como chegamos à quarta e última parte da rotina que faz Willie pular verticalmente, o conteúdo da posição 18261 não é incrementado. A instrução **CLR 18261** ajusta essa variável com 0; a ro-



tina de movimentação de Willie é, portanto, executada e verifica-se, pelo exame do teclado, se ele deve pular de novo.

A posição de Willie é carregada em X e colocada na pilha de máquina para armazenamento temporário. Como a figura tem apenas dois caracteres de altura quando está parada ou andando, e três, quando está pulando, o caractere mais alto tem que ser apagado de novo. Note que, agora, Willie está numa posição intermediária, usada para tornar o pulo mais suave.

U é carregado com 1536, que aponta para uma parte do céu no topo da tela. Quando é chamada, a rotina **CHARPR** apaga a cabeça da figura anterior.

A posição de Willie é recuperada da pilha e adicionada a 256, o que faz a figura se mover um caractere abaixo na tela. Lembre-se de que sua posição tinha sido decrementada com 256 no início desta rotina, quando Willie começou o salto. Assim, quando o novo valor é armazenado em 18261, o personagem simplesmente volta para a posição anterior ao do pulo.

U é então carregado com 17774, endereço inicial dos dados para a figura de Willie com as pernas juntas. **CHARPR** é chamada duas vezes e X é adicionado com 254, imprimindo a figura completa de Willie na tela.



```
10    org 54854
20    pl ld a,(-5202)
30    cp 1
40    jr nz,pb
50    inc a
60    ld (-5202),a
70    ld hl,(-5205)
80    ld de,32
90    sbc hl,de
100    ld (-5205),hl
110    ld de,(62407)
120    add hl,de
130    ld a,10
140    push hl
150    call 77
160    pop hl
170    ld de,32
180    add hl,de
190    ld a,11
200    push hl
210    call 77
220    pop hl
230    ld de,32
240    add hl,de
250    ld a,255
260    call 77
270    ret
280    pb cp 2
290    jr nz,pc
300    inc a
310    ld (-5202),a
320    ld hl,(-5205)
330    ld de,(62407)
340    add hl,de
350    ld a,0
360    push hl
370    call 77
380    pop hl
390    ld de,32
400    add hl,de
410    ld a,1
420    call 77
430    ret
440    pc cp 3
450    jr nz,pd
460    inc a
470    ld (-5202),a
480    ld hl,(-5205)
490    ld de,(62407)
500    add hl,de
510    ld a,10
520    push hl
530    call 77
540    pop hl
550    ld de,32
560    add hl,de
570    ld a,11
580    call 77
590    ret
600    pd cp 4
610    jr nz,mp
620    ld a,0
630    ld (-5202),a
640    ld hl,(-5205)
650    push hl
660    ld de,(62407)
670    add hl,de
680    ld a,255
690    call 77
700    pop hl
710    ld de,32
720    add hl,de
730    ld (-5205),hl
740    jp 54603
750    mp ret
760    end
```

SUBINDO

A rotina de salto é dividida em quatro partes; cada uma delas é executada separadamente quando a rotina de movimentação de Willie é chamada. No último artigo, vimos como o processador é direcionado para a rotina **pl** quando o conteúdo da posição de memória −5202 não é zero. Vimos também como examinar o teclado para descobrir se as teclas de controle M e N foram pressionadas, individualmente ou ao mesmo tempo.

Se a tecla M foi pressionada e Willie deve saltar, a posição de memória −5202 é carregada com 1. Isso fará com que o processador seja direcionado para esta rotina na próxima vez que a rotina principal do jogo chamar a rotina de movimentação.

Se M e N foram pressionadas, Willie deve saltar na diagonal e 129 é carregado em −5202. Este caso será examinado no próximo artigo.

Por enquanto, suponhamos que a rotina de movimentação detectou que a tecla N foi pressionada na última vez que foi chamada. Agora, ela voltou a ser chamada, o que fez o processador direcionar-se para a rotina do salto, cuja execução se inicia...

O conteúdo da posição −5202 da memória é carregado novamente no acumulador. Em seguida, o valor em A é comparado com 1. Como a suposição feita exige que o valor seja 1, a instrução seguinte, **jr nz,pb**, não é executada e o processador continua com **inc a**. Com isso, o conteúdo do acumulador passa a ser 2 — valor que é colocado de volta em −5202 pela instrução **ld (−5202), a**. Incrementa-se, assim, o contador de salto para que, na próxima vez que a rotina de movimentação for chamada, ela acione a rotina **pl**. Neste ponto, porém, a primeira parte do salto é deixada de lado e o processador executa a segunda.

A posição de Willie na tela é carregada dos enderecos −5205 e −5204 no par HL. O número 32 é subtraído dessa posição e colocado em DE; a instrução **sbc hl, de** é executada. Isso coloca o apontador de posição um caractere acima de Willie na tela. O resultado é carregado de volta nos endereços que contêm a posição de Willie, −5205 e −5204.



Embora esse valor tenha sido colocado nos endereços, ele permanece no par HL. Todos os **ld** são essencialmente operações de cópia, que colocam o valor na nova posição ou registro, mantendo esse valor na posição ou registro de origem. No nosso caso isso é importante, pois o valor em HL é somado ao endereço inicial da Tabela de Nomes da VRAM, passando a apontar para a posição de Willie na TN.

O acumulador é carregado com 10,

código do primeiro padrão da figura de Willie subindo, com as pernas abertas. O apontador no par HL é guardado na pilha e a rotina 77 da ROM é chamada. Essa rotina escreve o valor de A no endereço da VRAM apontado por HL — o que equivale, aqui, a colocar o padrão na tela.

O apontador é recuperado da pilha e somado com 32, voltando-se para uma posição abaixo da anterior. A é carregado com 11, código do segundo padrão da figura de Willie. O apontador é novamente guardado na pilha e a rotina 77 da ROM é chamada.

A figura de Willie saltando já está completa. Falta só apagar uma parte da figura anterior. Para isso, recupera-se o apontador da pilha e adiciona-se 32, fazendo-o apontar para o padrão que se pretende eliminar. O código do padrão de céu, 255, é colocado em A e a rotina 77 é chamada, apagando parte da figura anterior. Em seguida, o processador retorna.



WILLIE ESTÁ NO AR

A segunda parte da rotina de salto é chamada assim que se aciona a rotina de movimentação. Como as outras partes, ela começa verificando se a posição de memória −5202 contém 2 — o que acontece se esta parte ainda não foi executada.

Lembre-se de que o conteúdo de −5202 está no acumulador quando o processador aciona esta parte — portanto, não é necessário carregá-lo outra vez. Se o valor 2 está presente, o processador continua com a rotina e incrementa o contador de salto com as instruções **inc a** e **ld (−5202), a**. Quando esta parte for acessada na próxima vez, **cp 2** não irá encontrar o valor 2 e **jr nz,pc** mandará então o processador para a terceira parte do salto.

Se o valor 2 estiver presente e esta parte da rotina foi acionada, a atual posição de Willie é transferida dos endereços −5205 e −5204 para o par de registros HL. O par DE é carregado com o endereço inicial da TN da VRAM e somado em HL. O acumulador é carregado com 0, o código do primeiro padrão da figura de Willie parado. O apontador em HL é guardado na pilha e a rotina 77 da ROM é chamada.

O apontador é recuperado da pilha e somado com 32 por meio de DE, o que o faz se mover uma posição para baixo. A é carregado com 1, o código do segundo padrão da figura. A rotina 77 é chamada de novo. Neste ponto, Willie está impresso no ar com as pernas juntas. Observe que agora não é preciso apagar nada, pois a figura foi impressa sobre a anterior.

O processador retorna da rotina até a próxima chamada

DESCENDO

Desta vez, o processador executa a terceira parte da rotina de salto. Primeiro, ele verifica se é necessário acionar a quarta parte do salto.

Em seguida, o contador de salto, que ainda está no acumulador, é incrementado e devolvido a −5202, para que o processador passe para a quarta parte na próxima vez.

O apontador de Willie na tela é carregado no par HL. O par DE é carregado com o endereço inicial da TN da VRAM e somado em HL. Em seguida, a mesma figura da primeira parte é impressa de maneira idêntica, utilizando a rotina 77 da memória ROM. O processador retorna desta parte.

FIM DO SALTO

Mais uma vez a rotina inicia com um **cp**, verificando se esta é a parte necessária da rotina de salto. Para que o processador chegue até aqui, o conteúdo do acumulador deve ser 4 ou 129. Se for 4, esta é a última parte da rotina do salto; se for 129, Willie irá saltar na diagonal. Nesse caso, a instrução **jr nz,mp** manda o processador para a instrução de retorno do final deste programa. Essa instrução será apagada pela rotina de salto diagonal, que apresentaremos no próximo artigo de *Avalanche*.

Nenhum teste será feito, nem precisaremos incrementar o contador de salto. É necessário, porém, ajustar esse contador com zero, para que, quando a rotina de movimentação for chamada de novo, o processador execute a primeira parte, dada no artigo anterior, que verifica se alguma tecla foi pressionada. Isso é feito pelas instruções **ld a,0** e **ld (−5202),a**.

A posição de Willie é mais uma vez carregada no par HL, sendo armazenada na pilha. O endereço inicial da TN da VRAM é colocado em DE e somado ao valor de HL. O acumulador é carregado com 255, o código do padrão de céu. A rotina 77 da ROM é chamada, apagando parte da figura anterior.

A posição de Willie é recuperada da pilha para HL, somada com 32 através de DE e, finalmente, devolvida aos endereços −5205 e −5204.

A primeira rotina de movimentação de Willie é chamada por **jp 54603**. Não se preocupe com o que restou de sua figura, pois nossa rotina apagará todos os vestígios indesejáveis.

O último **ret** da listagem é colocado apenas para prevenir possíveis erros, que ocorreriam se a rotina de salto diagonal fosse chamada sem estar presente na memória. Essa instrução será apagada pela parte do programa dada no próximo artigo de *Avalanche*.

MODELOS DA REALIDADE

Para ganhar a quina da loto, basta escolher cinco números ao acaso — os cinco números certos, é claro. Veja aqui algumas dicas estatísticas para programar eventos aleatórios.



Os engenheiros constroem modelos em miniatura, para testar suas propriedades. Quando fazem isso, estão simulando um sistema. O programa do artigo da página 1121 executa algo bem parecido: ele simula um evento, oferecendo resultados compatíveis com a realidade. Veremos agora como combinar os princípios da simulação e da estatística para resolver problemas de ordem prática e para acrescentar a nossos jogos uma dose de realidade.

TIPOS DE MODELO

Um modelo é uma representação simplificada de um sistema real, por meio da qual se procura explicar ou prever as características e o comportamento desse sistema. Existem três tipos de modelo — icônico, análogo e simbólico —, nem todos úteis aos usuários de microcomputadores.

Modelos icônicos não possuem partes móveis. Maquetes de estradas e prédios incluem-se nessa categoria. Ao projetar a construção de uma ponte sobre um estuário, por exemplo, engenheiros civis providenciam uma maquete da ponte e da área adjacente, a fim de estudar os efeitos da maré, da correnteza e do vento. Embora se utilizem computadores para interpretar os resultados, a necessidade de coletar dados através de um experimento torna a simulação dispensável.

Modelos análogos representam um fenômeno — ou quantidade — por outro. É o caso de um economista que, para representar o fluxo de dinheiro na economia mundial, utiliza o fluxo de água de um conjunto de tanques cheios até diferentes níveis. Esse tipo de representação pode ser feito eletronicamente — empregando-se computadores analógicos e não digitais.

O terceiro tipo de modelo, o simbólico, é o que apresenta particular interesse para o usuário do micro.

INCERTEZAS

Num modelo simbólico, representa-se determinado sistema por meio de uma fórmula matemática. Para estudar a distância S percorrida por um carro que se desloca à velocidade constante V por um certo período de tempo T, um modelo adequado seria **S = V*T**. Essa equação é chamada *determinística*, pois não comporta nenhum elemento de incerteza ou acaso.

Existem, contudo, eventos que não podem ser previstos com tal grau de precisão. Temos um bom exemplo disso no programa apresentado no artigo da página 776, que simula o lançamento de uma moeda. Modelos que descrevem fenômenos que incluem um elemento de acaso são chamados *estocásticos*. Este é o tipo de evento que se costuma simular, especialmente em jogos.

Algumas variáveis só podem assumir valores discretos como 0, 1, 2, 3... O número de gols em uma partida de futebol pertence a essa categoria. Quando simulamos variáveis aleatórias como essa, precisamos sempre levar em conta que certos processos aparentemente diferentes têm, na realidade, a mesma estrutura.

Tomemos como exemplo o programa que simula o lançamento da moeda. Ele poderia ser facilmente modificado a fim de fornecer os resultados de um jogo de dados ou da cobrança de pênaltis. Em todas essas circunstâncias há um certo número de eventos independentes, cada qual com uma probabilidade definida. No caso de lançarmos uma moeda quatro vezes, vários resultados são possíveis: quatro caras, três caras e uma coroa, duas caras e duas coroas, uma cara e três coroas, quatro coroas. Cada um dos resultados, porém, tem uma probabilidade definida. Os processos desse tipo, conhecidos como *processos de Bernoulli*, são formados por variáveis alea-

- TIPOS DE MODELO
- INCERTEZAS
- PROBABILIDADE E COMPORTAMENTO ALEATÓRIO
- VARIÁVEIS ALEATÓRIAS
- SIMULAÇÕES
- DISTRIBUIÇÃO UNIFORME
- DISTRIBUIÇÃO NORMAL
- DISTRIBUIÇÃO EXPONENCIAL
- COMPARAÇÃO DE EVENTOS



tórias discretas. Eles não se prestam, por exemplo, para descrever o número de acidentes do campeonato mundial de Fórmula-1 ou o número de chamadas telefônicas que chegam a uma central em meia hora. Nessas situações, os eventos ocorrem ao acaso durante um certo período de tempo e não são o resultado de experimentos sucessivos. Tais processos são denominados *processos de Poisson*.

Para obter uma distribuição de Poisson, digite este programa:

```
10 BORDER 0: PAPER 0: INK 7:
CLS
20 POKE 23658,0
40 PRINT AT 1,6; INVERSE 1;"S
IMULACAO DE POISSON ": PRINT
: PRINT
50 INPUT "QUAL E O VALOR MEDI
O ";a
 70 INPUT "TAMANHO DA AMOSTRA
";n
 80 FOR i=1 TO n
 90 LET c=0: LET t=1
100 LET s=EXP (-a)
110 LET t=t*(RND*1)
120 IF t<=s THEN  PRINT "
";c;: GOTO 150
130 LET c=c+1
140 GOTO 110
150 NEXT i
160 INPUT " OUTRA AMOSTRA  (S/
N) ?";g$
170 IF g$="s" THEN  GOTO 10
180 STOP
```



```
10 R=RND(-TIME)
30 CLS:WIDTH 40
40 PRINT "Distribuão de Poisson
":PRINT:PRINT
50 INPUT "Qual é o valor da méd
ia";A
60 PRINT
70 INPUT "Tamanho da amostra";N
80 FOR I=1 TO N
90 C=0:T=1
100 S=EXP(-A)
110 T=T*RND(1)
120 IF T<=S THEN PRINT LEFT$(ST
R$(C)+"         ",8);:GOTO 150
130 C=C+1
140 GOTO 110
150 NEXT I
160 PRINT:INPUT "Outra amostra
(Y/N)";G$
170 IF G$="Y" THEN 30
180 END
```



```
30  HOME
40  PRINT  TAB( 9);"DISTRIBUICA
O DE POISSON": PRINT : PRINT
50  INPUT "QUAL O VALOR DA MEDI
A ";A
60  PRINT
70  INPUT "TAMANHO DA AMOSTRA "
;N
80  FOR I = 1 TO N
90 C = 0:T = 1
100 S =  EXP ( - A)
110 T = T *  RND (1)
120  IF T <  = S THEN  PRINT  L
EFT$ ( STR$ (C) + "         ",8);
: GOTO 150
130 C = C + 1
140  GOTO 110
150  NEXT I
160  PRINT : INPUT "OUTRA AMOST
RA (S/N) ";G$
170  IF G$ = "S" THEN 30
180  END
```



```
30 CLS
40 PRINT @6,"simulacao de poiss
on":PRINT:PRINT
50 INPUT"QUAL E O VALOR MEDIO "
;A
60 PRINT
70 INPUT"TAMANHO DA AMOSTRA ";N
80 FOR I=1 TO N
90 C=0:T=1
100 S=EXP(-A)
110 T=T*RND(0)
120 IF T<=S THEN PRINT LEFT$(ST
R$(C)+"         ",8);:GOTO 150
130 C=C+1
140 GOTO 110
150 NEXT I
160 PRINT:INPUT"OUTRA AMOSTRA (
S/N) ";G$
170 IF G$="S" THEN 30
180 END
```

Ao ser executado, o programa solicita o valor médio, bem como o número de elementos da amostra. A parte principal do programa (linhas 80 a 150) usa uma equação para gerar as variáveis de Poisson. A variável **S** recebe o valor de **e** (uma constante matemática) elevada ao valor de **A** (valor da média). A linha 110 escolhe um número aleatório entre **0** e **T**. A linha 120 compara as variáveis **S** e **T** para decidir se imprime ou não uma variável **C**.

Suponhamos que você queira criar um jogo onde uma nave espacial enfrenta uma chuva de meteoritos. Se o número médio de "colisões" em um período de um minuto é dois, e você precisa de cinco simulações de um minuto, o computador poderá fornecer o resultado 2, 3, 2, 1, 0 como amostra do número de choques que a nave sofrerá em um minuto. Cada valor não é muito diferente de 2 (a média) e há cinco valores — o tamanho da amostra.

Poderíamos usar a mesma técnica para simular os resultados de uma série de partidas de futebol? É claro que sim. Estaremos lidando com uma variável discreta, pois o número de gols em uma partida só pode assumir valores inteiros. Gols são eventos que ocorrem ao acaso, em determinado espaço de tempo — e não o resultado de um certo número

de experimentos repetidos. Portanto, nesse caso, um processo de Poisson é adequado.

Além do número de partidas, precisaremos escolher a média de gols por partida. Este é o momento de buscar dados no mundo real. A tabela abaixo mostra a média de gols por partida nos campeonatos da primeira divisão inglesa entre 1977 e 1983.

Primeira divisão
Média de gols por partida

Temporada	Time da casa	Time visitante
77-78	1.6039	1.0606
78-79	1.5498	1.0844
79 80	1.6925	0.9481
80-81	1.6364	1.0216
81 82	1.4545	1 0844
82-83	1.7532	0.9822

Baseados nesses valores, podemos estimular uma média de 1.7 gols por partida para o time da casa e de 1.0 para o time visitante. Dividindo esses números por dois, obteremos as médias de 0.85 e 0.5 para meio tempo de jogo. De posse desses dados, fica fácil elaborar o programa:

```
20 DIM a$(25,10): DIM h$(25,
10): DIM h(50): DIM a(50):
DIM f(50): DIM g(50)
30 BORDER 0: PAPER 0: INK 7:
CLS
50 PRINT AT 0,8; INVERSE 1;"
 PLACAR FINAL  "
70 INPUT " QUANTAS PARTIDAS (
1-25)? ";n
75 IF n<1 OR n>25 THEN  GOTO
 70
80 FOR i=1 TO n
90 PRINT "PARTIDA";i
100 INPUT "TIME DA CASA";h$(i)
110 INPUT "TIME VISITANTE";a$(
i)
120 NEXT i
130 PAUSE 400
140 PRINT
150 FOR i=1 TO 2*n
160 LET c=0: LET t=1
170 LET s=EXP (-.85)
180 LET t=t*(RND*1)
190 IF t<=s THEN  LET h(i)=c:
GOTO 220
200 LET c=c+1
210 GOTO 180
220 NEXT i
230 FOR i=1 TO 2*n
240 LET c=0: LET t=1
250 LET s=EXP (-.5)
260 LET t=t*(RND*1)
270 IF t<=s THEN  LET a(i)=c:
GOTO 300
280 LET c=c+1
290 GOTO 260
300 NEXT i
310 CLS
320 PRINT TAB (7);"PLACAR PRIM
EIRO TEMPO";''
330 FOR i=1 TO n
340 PRINT h$(i);h(i);TAB 20;a$
(i);a(i): PRINT
350 NEXT i
360 PRINT " QUALQUER TECLA PAR
A CONTINUAR"
370 IF INKEY$="" THEN  GOTO
 370
375 CLS
380 PRINT TAB 12;"PLACAR FINAL
"''
390 FOR i=1 TO n
400 LET f(i)=h(i)+h(n+i)
410 LET g(i)=a(i)+a(i+n)
420 PRINT h$(i);f(i);TAB 20;a$
(i);g(i): PRINT
430 NEXT i
440 INPUT " NOVAMENTE ?   (s/n
) ";g$
450 IF g$="n" THEN  GOTO 490
460 INPUT "MESMOS TIMES (s/n)
?";p$
470 IF p$="n" THEN  GOTO 70
480 GOTO 150
490 STOP
```

```
5 R=RND(-TIME)
20 DIM A$(25),H$(25),H(50),A(50
),FA(25),FH(25)
30 CLS:WIDTH 40
50 PRINT TAB(13);"SCORE FINAL"
60 PRINT:PRINT
70 INPUT"Quantas partidas (1-25
)";N
75 IF N<1 OR N>25 THEN 70
80 FOR I=1 TO N
90 PRINT "PALITO";I
100 INPUT "Time da casa";H$(I)
110 INPUT "Time de fora";A$(I)
120 NEXT I
130 FOR V=1 TO 2000:NEXT V
140 PRINT
150 FOR I=1 TO 2*N
160 C=0:T=1
170 S=EXP(-.85)
180 T=T*RND(0)
190 IF T<=S THEN H(I)=C:GOTO 22
0
200 C=C+1
210 GOTO 180
220 NEXT I
230 FOR I=1 TO 2*N
240 C=0:T=1
250 S=EXP(-.5)
260 T=T*RND(0)
270 IF T<=S THEN A(I)=C:GOTO 30
0
280 C=C+1
290 GOTO 260
300 NEXT I
310 CLS
320 PRINT TAB(9);"Score temporá
rio":PRINT:PRINT
330 FOR I=1 TO N
340 PRINT H$(I);H(I),A$(I);A(I)
350 NEXT I
360 P..INT "Aperte qualquer tecl
a"
370 IF INKEY$="" THEN 370
380 CLS
390 FOR I=1 TO N
400 FH(I)=H(I)+H(N+I)
410 FA(I)=A(I)+A(N+I)
420 PRINT H$(I);FH(I),A$(I);FA(
I)
430 NEXT
440 PRINT INPUT"OUTRA VEZ (S/N)
";G$
450 IF G$="N" THEN END
460 INPUT "Mesmos times (S/N)";
P$
470 IF P$="N" THEN 70
480 GOTO 150
```



```
20  DIM A$(25),H$(25),H(50),A(5
0),FA(25),FH(25)
30  HOME
50  PRINT  TAB( 13);"SCORE FINA
L "
60  PRINT : PRINT
70  INPUT "QUANTAS PARTIDAS (1-
25) ";N
75  IF N < 1 OR N > 25 THEN 70
80  FOR I = 1 TO N
90  PRINT "PARTIDA ";I
100  INPUT "TIME DA CASA ";H$(I
)
110  INPUT "TIME DE FORA ";A$(I
)
120  NEXT I
130  FOR V = 1 TO 2000: NEXT V
140  PRINT
150  FOR I = 1 TO 2 * N
160 C = 0:T = 1
170 S =  EXP ( - .85)
180 T = T *  RND (1)
190  IF TT <  = S THEN H(I) = C
: GOTO 220
200 C = C + 1
210  GOTO 180
220  NEXT I
230  FOR I = 1 TO 2 * N
240 C = 0:T = 1
250 S =  EXP ( - .5)
260 T = T *  RND (1)
270  IF T <  = S THEN A(I) = C:
 GOTO 300
280 C = C + 1
290  GOTO 260
300  NEXT I
310  HOME
320  PRINT "SCORE PARCIAL ": PR
INT : PRINT
330  FOR I = 1 TO N
340  PRINT H$(I);H(I),A$(I);A(I
)
350  NEXT I
360  PRINT "APERTE QUALQUER TEC
LA "
370  GET G$
375  HOME
380  PRINT  TAB( 13);"SCORE FIN
AL": PRINT : PRINT
390  FOR I = 1 TO N
400 FH(I) = H(I) + H(N + I)
410 FA(I) = A(I) + A(N + I)
420  PRINT H$(I);FH(I),A$(I);FA
(I)
```

```
430   NEXT
440   PRINT : INPUT "OUTRA VEZ (
S/N) ";P$
450   IF P$ = "N" THEN  END
460   INPUT "MESMOS TIMES (S/N)
";G$
470   IF G$ = "N" THEN 70
480   GOTO 150
```



```
20 DIM A$(25),H$(25),H(50),A(50
),FA(25),FH(25)
30 CLS
50 PRINT @11,"placar final"
60 PRINT:PRINT
70 INPUT"QUANTAS PARTIDAS (1-25
) ";N
80 FOR I=1 TO N
90 PRINT"PARTIDA ";I
100 INPUT"TIME DA CASA ";H$(I)
110 INPUT"VISITANTE ";A$(I):PRI
NT
120 NEXT I
130 FOR V=1 TO 2000:NEXT V
140 PRINT
150 FOR I=1 TO 2*N
160 C=0:T=1
170 S=EXP(-.85)
180 T=T*RND(0)
190 IF T<=S THEN H(I)=C:GOTO220
200 C=C+1
210 GOTO 180
220 NEXT I
230 FOR I=1 TO 2*N
240 C=0:T=1
250 S=EXP(-.5)
260 T=T*RND(0)
270 IF T<=S THEN A(I)=C:GOTO300
280 C=C+1
290 GOTO 260
300 NEXT I
310 CLS
320 PRINT @5,"placar primeiro t
empo":PRINT:PRINT
330 FOR I=1 TO N
340 PRINT H$(I),H(I),A$(I);A(I)
350 NEXT I
360 PRINT" QUALQUER TECLA PARA
CONTINUAR"
370 IF INKEY$="" THEN 370
375 CLS
380 PRINT @10,"placar final":PR
INT:PRINT
390 FOR I=1 TO N
400 FH(I)=H(I)+H(N+I)
410 FA(I)=A(I)+A(N+I)
420 PRINT H$(I);FH(I),A$(I),FA(
I)
430 NEXT
440 PRINT:INPUT"NOVAMENTE  (S/N
) ",G$
450 IF G$="N" THEN END
460 INPUT"MESMOS TIMES  (S/N) "
;P$
470 IF P$="N" THEN 70
480 GOTO 150
```

Da linha 20 a 120 o programa solicita as informações iniciais. Para evitar muita digitação, podemos substituir os nomes dos times por letras. As linhas 150 a 300 usam o algoritmo de Poisson, idêntico ao do programa anterior, para gerar os resultados parciais e finais dos jogos. As linhas 320 a 350 cuidam da saída dos resultados parciais e as linhas 390 a 430, dos resultados finais.

A simulação de Poisson funciona bem com variáveis discretas. Algumas vezes, porém, precisamos utilizar variáveis fracionárias ou contínuas. Em uma simulação de competição de salto em distância, por exemplo, podemos obter uma medida qualquer entre sete e nove metros — o comprimento do salto é uma variável aleatória, contínua.

Embora a função **RND** seja adequada para simulações de Poisson com números inteiros, será necessário modificá-la um pouco para utilizá-la com variáveis contínuas. O quadro da página 1180 compara as distribuições obtidas com os dois tipos de variáveis: variáveis discretas resultam em distribuições *uniformes*, enquanto variáveis contínuas resultam em distribuições *exponenciais* e distribuições *normais*.

Digite o próximo programa para transformar os números aleatórios uniformemente distribuídos, criados pela função **RND** do computador, em números distribuídos exponencialmente.



```
10 BORDER 0: INK 7: PAPER 0:
CLS
20 POKE 23658,0
30 PRINT AT 0,4; INVERSE 1;"
SIMULACAO EXPONENCIAL "''
50 INPUT "QUAL E O VALOR MEDI
O ? ";a
70 INPUT "TAMANHO DA AMOSTRA
? ";n
80 FOR i=1 TO n
90 LET x=(-a)*LN (RND*1): LET
y=INT (x*10): PRINT "        ";
.1*y,
100 NEXT i
110 INPUT " OUTRA SIMULACAO (s
/n) ? ";g$
120 IF g$="s" THEN  GOTO 30
130 STOP
```



```
10 R=RND(-TIME)
30 CLS:WIDTH 40
40 PRINT TAB(8);"Simulação expo
nencial":PRINT:PRINT
50 INPUT "Qual o valor da média
";A
60 PRINT
70 INPUT "Tamanho da amostra";N
80 FOR I=1 TO N
90 X=(-A)*LOG(RND(1)):Y=INT(X*1
0):PRINT LEFT$(STR$(Y/10)+"
  ",8);
100 NEXT I
110 PRINT:INPUT "Outra vez (S/N
)";G$
120 IF G$="S" THEN 30
130 END
```



```
30   HOME
40   PRINT  TAB( 7);"SIMULACAO E
XPONENCIAL": PRINT : PRINT
50   INPUT "QUAL O VALOR DA MEDI
A ";A
60   PRINT
70   INPUT "TAMANHO DA AMOSTRA "
;N
80   FOR I = 1 TO N
90 X = ( - A) *  LOG ( RND (1))
:Y =  INT (X * 10): PRINT  LEFT
$ ( STR$ (Y / 10) + "         ",8
);
100   NEXT I
110   PRINT : INPUT "OUTRA SIMUL
ACAO (S/N) ";G$
120   IF G$ = "S" THEN 30
130   END
```



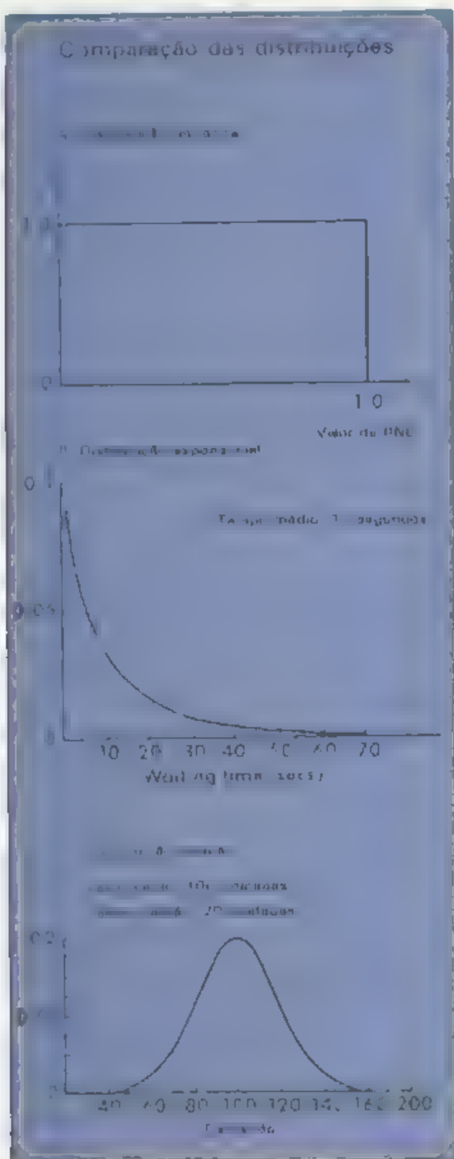
```
30 CLS
40 PRINT @5,"simulacao exponenc
ial":PRINT:PRINT
50 INPUT"QUAL E O VALOR MEDIO "
;A
60 PRINT
70 INPUT"TAMANHO DA AMOSTRA ",N
80 FOR I=1 TO N
90 X=(-A)*LOG(RND(0)):Y=INT(X*1
0):PRINT LEFT$(STR$(Y/10)+"
  ",8),
100 NEXT I
110 PRINT:INPUT"OUTRA SIMULACAO
  (S/N) ",G$
120 IF G$="S" THEN 30
130 END
```

Devemos informar ao programa, assim que o executamos, o valor da média. Feito isso, os valores da amostra são exibidos. A linha 90 realiza toda a tarefa, utilizando um recurso matemático chamado *método da transformação inversa*, para mudar a distribuição das variáveis. Esse programa pode ser usado para simular, por exemplo, o tempo de espera em um lava-rápido ou a duração das reservas de combustível em um jogo espacial. Em ambos os casos, o tempo decorrido é o elemento central.

DISTRIBUIÇÃO NORMAL

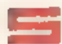
A curva normal é o tipo mais conhecido de distribuição estatística. Tem numerosas aplicações, descrevendo toda espécie de fenômeno natural — como a estatura e o peso de homens adultos. Além disso, constitui o meio mais apropriado para a representação de uma distribuição de erros.

Como exemplo do uso da distribuição normal em um modelo, consideremos esta situação: as vendas (S) de uma



empresa dependem de um valor-base (So) e do total gasto em propaganda (A). A relação poderia ser resumida na expressão S = So + (B*A) + desvio normal, onde B é uma constante conhecida.

Digite o programa a seguir para ver como funciona.



```
10 BORDER 0· PAPER 0· INK 7·
CLS
20 DIM u(50)· DIM y(50): DIM
z(50)
50 PRINT AT 0.7: INVERSE 1:"
SIMULACAO NORMAL "''
60 INPUT " QUAL E O VALOR MED
IO ? ";a
70 INPUT " QUAL E O VALOR MIN
IMO ? ";m
80 INPUT " TAMANHO DA AMOSTRA
 ? ";n
90 LET b=(a-m)/2.5
100 FOR i=1 TO n
110 LET t=0
120 FOR j=1 TO 15
130 LET y(j)=RND*1
140 LET t=t+y(j)· NEXT j
150 LET u(i)=((t/15)-.5)/SQR (
1/(12*15))
160 LET z(i)=a+(b*u(i))
170 LET p=INT (10*z(i)): PRINT
"     ";.1*p,
180 NEXT i
190 INPUT " OUTRA AMOSTRA ? ";
g$
200 IF g$="s" THEN  GOTO 50
210 STOP
```



```
10 R=RND(-TIME)
30 CLS:WIDTH 40
40 DIM U(50),Y(50),Z(50)
50 PRINT "Simulação normal":PRI
NT·PRINT
60 INPUT "Qual o valor da média
";A
70 INPUT "Qual o valor mínimo";
M
80 INPUT "Tamanho da amostra";N
90 B=(A-M)/2.5
100 FOR I=1 TO N
110 T=0
120 FOR J=1 TO 15
130 Y(J)=RND(1)
140 T=T+Y(J)·NEXT J
150 U(I)=((T/15)-.5)/SQR(1/(12*
15))
160 Z(I)=A+(B*U(I))
170 P=INT(10*Z(I)):PRINT LEFT$(
STR$(P/10)+"         ",8);
180 NEXT I
190 PRINT·INPUT"Outra amostra";
G$
200 IF G$="S" THEN 50
210 END
```



```
30  HOME
40  DIM U(50),Y(50),Z(50)
50  PRINT  TAB( 9);"DISTRIBUICA
O NORMAL": PRINT : PRINT
60  INPUT "QUAL O VALOR DA MEDI
A ";A
70  INPUT "QUAL O VALOR MINIMO
";M
80  INPUT "TAMANHO DA AMOSTRA "
;N
90 B = (A - M) / 2.5
100  FOR I = 1 TO N
110 T = 0
120  FOR J = 1 TO 15
130 Y(J) =  RND (1)
140 T = T + Y(J): NEXT J
150 U(I) = ((T / 15) - .5) /  S
QR (1 / (12 * 15))
160 Z(I) = A + (B * U(I))
170 P =  INT (10 * Z(I)): PRINT
  LEFT$ ( STR$ (P / 10) + "
   ",8);
180  NEXT I
190  PRINT : INPUT "OUTRA  AMOS
TRA (S/N) ";G$
200  IF G$ = "S" THEN 50
210  END
```



```
30 CLS
40 DIM U(50),Y(50),Z(50)
50 PRINT @7,"distribuicao norma
l" PRINT PRINT
60 INPUT"QUAL E O VALOR MEDIO "
;A
70 INPUT"QUAL E O VALOR MINIMO
",M
80 INPUT"TAMANHO DA AMOSTRA ",N
90 B=(A M)/2.5
100 FOR I=1 TO N
110 T=0
120 FOR J=1 TO 15
130 Y(J)=RND(0)
140 T=T+Y(J) NEXT J
150 U(I)=((T/15)  5) SQR(1/(12*
15))
160 Z(I)=A+(B*U(I))
170 P=INT(10*Z(I)  PRINT LEFT$
STR$(P/10)+'          ",8),
180 NEXT I
190 PRINT INPUT"OUTRA AMOSTRA
,G$
200 IF G$="S" THEN 50
210 END
```

Execute o programa e forneça o valor médio e o valor mínimo, bem como o tamanho da amostra. Tecnicamente, o intervalo de uma distribuição normal é infinito, o que deixa uma chance muito pequena de se obter um valor menor que o mínimo especificado.

As linhas 120 a 150 usam a função **RND** para criar quinze números entre 0 e 1, somando-os em seguida. A linha 160 calcula a média e o fator de escala. Depois de ter sido escalonado, o resultado é impresso.

Para maior clareza, poderá ser útil criar duas distribuições com a mesma média e aproximadamente o mesmo intervalo, mas com formatos diferentes. Execute o programa e forneça os valores 100 para a média, 50 para o mínimo e 40 para o tamanho da amostra. Se os valores obtidos fossem plotados, teríamos um gráfico semelhante ao da figura C (quadro da página 1180).

Agora, apague a linha 90 e as linhas 120 a 160. Substitua a linha 110 pela seguinte:

```
110 Z(I)=M+RND(1)*2*(A-M)
```

No TRS-Color, use **RND (0)** no lugar de **RND (1)**.

Com essas alterações, a distribuição normal é convertida em distribuição uniforme. Execute o programa novamente e entre os valores especificados, comparando os resultados. Desta vez, os valores levariam a um gráfico como o da figura A (página 1180).

Observe que a simulação normal acumula maior número de resultados próximos à média.

MODELOS E SIMULAÇÃO

- COMO SIMULAR O CLIMA
- VARIÁVEIS DE DISTRIBUIÇÃO NORMAL
- CONDIÇÕES DE MERCADO
- CONTROLE DE CAIXA

Retomando nossos estudos sobre o uso de modelos, apresentamos neste artigo um programa que se vale de alguns princípios econômicos para simular o balanço de uma empresa comercial.



No artigo da página 1176, examinamos o papel da simulação em várias áreas de pesquisa. Vimos também como gerar diferentes variáveis aleatórias, cada qual adequada a uma determinada situação. Como demonstrou nosso programa de simulação normal, a geração de variáveis normalmente distribuídas é muito simples, mas não tão eficiente — foram necessários quinze números aleatórios para criar uma única variável. O programa deste artigo usa um método mais eficaz, que permite simular diversos aspectos de uma pequena empresa comercial. Este programa também pode ser utilizado como um jogo (até bem divertido), mas constitui, de fato, um modelo de uma situação real.

S

```
 10 POKE 23658,8: POKE 23609,
 12
 20 PAPER 0: BORDER 0: INK 7:
 CLS
 30 PRINT AT 0,8; INVERSE 1;"
 BATATA QUENTE "
 40 DIM A$(4,12): DIM L(2):
 DIM D(10): DIM W(2): DIM Q(2)
 : DIM P(2)
 50 LET A$(1)="QUENTE E SECO":
 LET D(3)=150: LET D(4)=300
 60 LET A$(2)="QUENTE E UMIDO"
 : LET D(5)=100: LET D(6)=200
 70 LET A$(3)="FRIO E SECO":
 LET D(7)=250: LET D(8)=160
 80 LET A$(4)="FRIO E UMIDO":
 LET D(9)=200: LET D(10)=100
580 DIM C(4,2): LET C(1,1)= 1:
LET C(1,2)=.15
590 LET C(2,1)=0.5: LET C(2,2)
=0.25
600 LET C(3,1)=0.01: LET C(3,2
)=0.12
610 LET C(4,1)=10: LET C(4,2)=
10
620 INPUT "QUANTOS JOGADORES (
1-6) ? ";N
625 IF N<1 OR N>6 THEN  GOTO
620
```

```
627 DIM K(N): DIM T(2,N): DIM
O(2,N)
630 FOR I=1 TO 2: FOR J=1 TO N
640 LET T(I,J)=0
650 NEXT J: NEXT I
700 FOR K=1 TO 10
710 LET P1=INT (10*(.3+(RND*1/
2)))/10
720 LET P2=INT (10*(.2+(RND*1/
2)))/10
730 PRINT  INK 4; BRIGHT 1;"--
----------------------------
 Dia:"; INK 7;K; INK 4;'"-----
----------------------------"''
740 PRINT "Prob. de um dia que
nte e seco : ";100*P1;"%"
750 PRINT "Prob  de um dia sec
o :          ";100*P2;"%"
760 GOSUB 1400
770 LET U1=RND*1: LET U2=RND*1
780 LET V1=SQR (2*(LN (1/U1)))
790 LET V2=COS (2*PI*U2): LET
V3=SIN (2*PI*U2)
800 LET Z1=INT (V1*V2): LET Z2
=INT (V1*V3)
810 LET A1=P1*P2: LET A2=P1
820 LET A3=P1+P2-A1: LET A4=1:
LET F=RND*1
821 PRINT : IF F<A1 THEN  LET
R=1
822 IF F>A1 AND F<=A2 THEN
LET R=2
823 IF F>A2 AND F<=A3 THEN
LET R=3
824 IF F>A3 AND F<=A4 THEN
LET R=4
830 CLS : PRINT "O TEMPO ESTA
";A$(R)
840 LET D(1)=INT (D(1+R*2)+Z1*
25): LET D(2)=INT (D(2+R*2)+Z2
*40)
850 PRINT "Demanda de batatas
assadas =";D(1)
860 PRINT "Demanda de latas de
coca =";D(2)
 990 PRINT ''
1000 PRINT  INK 5; INVERSE 1;"J
OGADOR  GANHO  CUSTOS  LUCRO  "
1010 GOSUB 1600
1020 NEXT K
1030 PAUSE 200
1090 CLS
1100 PRINT "  RESULTADO FINAL A
POS 10 DIAS "''
1110 PRINT "JOGADOR","LUCRO TOT
AL"
1120 FOR J=1 TO N
1130 PRINT J,K(J): NEXT J
1140 PRINT '': PRINT "FIM DE JO
GO"
1150 STOP
1400 PRINT "SEU PEDIDO POR FAVO
R ": PRINT
1410 FOR J=1 TO N
1420 PRINT "JOGADOR";J: PRINT
1430 INPUT "NUMERO DE BATATAS Q
UENTES ";O(1,J)
1440 INPUT "NUMERO DE LATAS DE
COCA COLA";O(2,J)
1450 NEXT J
1460 RETURN
1600 FOR J=1 TO N
1610 FOR I=1 TO 2
1620 LET L=O(I,J)
1630 IF D(I)<L THEN  LET L=D(I)
1650 LET W(I)=C(2,I)*L
1670 LET Q(I)=C(1,I)*O(I,J)
1680 IF D(I)>L(I) THEN  GOTO 17
00
1690 LET Q(I)=Q(I)-C(3,I)*(O(I,
J)-D(I))
1700 LET P(I)=W(I)-Q(I)
1710 LET T(I,J)=T(I,J)+P(I)
1720 NEXT I
1730 LET K(J)=T(1,J)+T(2,J)-200
1740 LET E=W(1)+W(2)
1750 LET C=Q(1)+Q(2)+20
1760 LET P=P(1)+P(2)-20
1770 PRINT  INK 6;TAB 3;J;TAB 9
;E;TAB 18;C;TAB 25;P;''
1780 NEXT J
1790 RETURN
```





```
5 R=RND(-TIME)
10 PI=4*ATN(1)
20 CLS
30 PRINT TAB(13);"Batata quente
":PRINT:PRINT
580 C1(1)=.1:C1(2)=.15
590 C2(1)=.5:C2(2)=.25
600 C3(1)=.01:C3(2)=.12
620 INPUT"QUANTOS JOGADORES (1-
6) ";N
625 IF N<1 OR N>6 THEN 620
630 FOR I=1 TO 2:FOR J=1 TO N
640 TP(I,J)=0
650 NEXT J,I
700 FOR K=1 TO 10
710 P1=INT(10*(.3+RND(1)/2))/10
720 P2=INT(10*(.2+RND(1)/2))/10
725 PRINT" QUALQUER TECLA PARA
CONTINUAR"
726 IF INKEY$="" THEN 726
730 PRINT"dia";K:PRINT
740 PRINT"PROB. DE UM DIA QUENT
E :";100*P1;"%"
750 PRINT"PROB. DE UM DIA SECO
:";100*P2;"%"
760 GOSUB 1400
770 U1=RND(1):U2=RND(1)
780 V1=SQR(2*(LOG(1/U1)))
790 V2=COS(2*PI*U2):V3=SIN(2*PI
*U2)
800 Z1=INT(V1*V2):Z2=INT(V1*V3)
810 A1=P1*P2:A2=P1
820 A3=P1+P2-A1:A4=1:F=RND(1)
```

```
821 CLS:IF F<=A1 THEN 830
822 IF F>A1 AND F<=A2 THEN 870
823 IF F>A2 AND F<=A3 THEN 810
824 IF F>A3 AND F<=A4 THEN 950
830 PRINT"O TEMPO ESTA QUENTE E
 SECO"
840 D(1)=150+Z1*25:D(2)=300+Z2*
40:GOTO 970
870 PRINT"O TEMPO ESTA QUENTE E
 UMIDO"
880 D(1)=100+Z1*25:D(2)=200+Z2*
40:GOTO 970
910 PRINT"O TEMPO ESTA FRIO E S
ECO"
920 D(1)=250+Z1*25:D(2)=160+Z2*
40:GOTO 970
950 PRINT"O TEMPO ESTA FRIO E U
MIDO"
960 D(1)=200+Z1*25:D(2)=100+Z2*
40
970 PRINT"DEMANDA DE BATATAS AS
SADAS=";D(1)
980 PRINT"DEMANDA DE LATAS DE C
OCA=";D(2)
1000 PRINT"JOGADOR  GANHO  CUST
OS  LUCRO"
1010 GOSUB 1600
1020 NEXT K
1030 FOR I=1 TO 2000:NEXT
1090 CLS
1100 PRINT" RESULTADO FINAL APO
S 10 DIAS ":PRINT:PRINT
1110 PRINT"JOGADOR","LUCRO TOTA
L"
1120 FOR J=1 TO N
1130 PRINT J,TT(J):NEXT J
1140 PRINT:PRINT:PRINT"FIM DE J
OGO"
1150 END
1400 PRINT:PRINTTAB(5);"seu ped
ido por favor":PRINT
1410 FOR J=1 TO N
1420 PRINT"JOGADOR";J:PRINT
1430 INPUT"NUMERO DE BATATAS";O
(1,J)
1440 INPUT"NUMERO DE LATAS DE C
OCA COLA";O(2,J)
1450 NEXT J
1460 RETURN
1600 FOR J=1 TO N
1610 FOR I=1 TO 2
1620 L=O(I,J)
1630 IF D(I)<L THEN L=D(I)
1650 RV(I)=C2(I)*L
1670 TC(I)=C1(I)*O(I,J)
1680 IF D(I)<=L THEN RC(I)=TC(I
)-C3(I)*(O(I,J)-D(I))
1700 P(I)=RV(I)-TC(I)
1710 TP(I,J)=TP(I,J)+P(I)
1720 NEXT I
1730 TT(J)=TP(1,J)+TP(2,J)-200
1740 E=RV(1)+RV(2)
1750 C=TC(1)+TC(2)+20
1760 P=P(1)+P(2)-20
1770 PRINTUSING"    #     ###.##
####.## ###.##";J,E,C,P
1780 NEXT J
1790 RETURN
```



```
10 PI=4*ATN(1)
20 HOME
30 PRINT TAB(13);"BATATA QUENTE
":PRINT:PRINT
580 C1(1)=.1:C1(2)=.15
590 C2(1)=.5:C2(2)=.25
600 C3(1)=.01:C3(2)=.12
620 INPUT"QUANTOS JOGADORES (1-
6) ";N
625 IF N<1 OR N>6 THEN 620
630 FOR I=1 TO 2:FOR J=1 TO N
640 TP(I,J)=0
650 NEXT J,I
700 FOR K=1 TO 10
710 P1=INT(10*(.3+RND(1)/2))/10
720 P2=INT(10*(.2+RND(1)/2))/10
725 PRINT" QUALQUER TECLA PARA
CONTINUAR"
726 GET W$
730 PRINT"DIA";K:PRINT
740 PRINT"PROB. DE UM DIA QUENT
E :";100*P1;"%"
750 PRINT"PROB. DE UM DIA SECO
:";100*P2;"%"
760 GOSUB 1400
770 U1=RND(1):U2=RND(1)
780 V1=SQR(2*(LOG(1/U1)))
790 V2=COS(2*PI*U2):V3=SIN(2*PI
*U2)
800 Z1=INT(V1*V2):Z2=INT(V1*V3)
810 A1=P1*P2:A2=P1
820 A3=P1+P2-A1:A4=1:F=RND(1)
821 HOME:IF F<=A1 THEN 830
822 IF F>A1 AND F<=A2 THEN 870
823 IF F>A2 AND F<=A3 THEN 810
824 IF F>A3 AND F<=A4 THEN 950
830 PRINT"O TEMPO ESTA QUENTE E
 SECO"
840 D(1)=150+Z1*25:D(2)=300+Z2*
40:GOTO 970
870 PRINT"O TEMPO ESTA QUENTE E
 UMIDO"
880 D(1)=100+Z1*25:D(2)=200+Z2*
40:GOTO 970
910 PRINT"O TEMPO ESTA FRIO E S
ECO"
920 D(1)=250+Z1*25:D(2)=160+Z2*
40:GOTO 970
950 PRINT"O TEMPO ESTA FRIO E U
MIDO"
960 D(1)=200+Z1*25:D(2)=100+Z2*
40
970 PRINT"DEMANDA DE BATATAS AS
SADAS=";D(1)
980 PRINT"DEMANDA DE LATAS DE C
OCA=";D(2)
1000 PRINT"JOGADOR  GANHO  CUST
OS  LUCRO"
1010 GOSUB 1600
1020 NEXT K
1030 FOR I=1 TO 2000:NEXT
1090 HOME
1100 PRINT" RESULTADO FINAL APO
S 10 DIAS ":PRINT:PRINT
1110 PRINT"JOGADOR","LUCRO TOTA
L"
1120 FOR J=1 TO N
1130 PRINT J,TT(J):NEXT J
1140 PRINT:PRINT:PRINT"FIM DE J
OGO"
1150 END
1400 PRINT:PRINTTAB(5);"SEU PED
IDO POR FAVOR":PRINT
1410 FOR J=1 TO N
1420 PRINT"JOGADOR";J:PRINT
1430 INPUT"NUMERO DE BATATAS";O
(1,J)
1440 INPUT"NUMERO DE LATAS DE C
OCA COLA";O(2,J)
1450 NEXT J
1460 RETURN
1600 FOR J=1 TO N
1610 FOR I=1 TO 2
1620 L=O(I,J)
1630 IF D(I)<L THEN L=D(I)
1650 RV(I)=C2(I)*L
1670 TC(I)=C1(I)*O(I,J)
1680 IF D(I)<=L THEN RC(I)=TC(I
)-C3(I)*(O(I,J)-D(I))
1700 P(I)=RV(I)-TC(I)
1710 TP(I,J)=TP(I,J)+P(I)
1720 NEXT I
1730 TT(J)=TP(1,J)+TP(2,J)-200
1740 E=RV(1)+RV(2)
1750 C=TC(1)+TC(2)+20
1760 P=P(1)+P(2)-20
1770 PRINT TAB(2);J;TAB(9);E;TA
B(17);C;TAB(25);P
1780 NEXT J
1790 RETURN
```



```
10 PI=4*ATN(1)
20 CLS
30 PRINT @10,"batata quente":PR
INT PRINT
580 C1(1)=.1:C1(2)=.15
590 C2(1)=.5:C2(2)=.25
600 C3(1)=.01:C3(2)=.12
620 INPUT"QUANTOS JOGADORES (1-
6) ";N
625 IF N<1 OR N>6 THEN 620
630 FOR I=1 TO 2:FOR J=1 TO N
640 TP(I,J)=0
650 NEXT J,I
700 FOR K=1 TO 10
710 P1=INT(10*(.3+RND(0)/2))/10
720 P2=INT(10*(.2+RND(0)/2))/10
725 PRINT" QUALQUER TECLA PARA
CONTINUAR"
726 IF INKEY$="" THEN 726
730 PRINT"dia";K:PRINT
740 PRINT"PROB. DE UM DIA QUENT
E :";100*P1;"%"
750 PRINT"PROB. DE UM DIA SECO
:";100*P2;"%"
760 GOSUB 1400
770 U1=RND(0):U2=RND(0)
780 V1=SQR(2*(LOG(1/U1)))
790 V2=COS(2*PI*U2):V3=SIN(2*PI
*U2)
800 Z1=INT(V1*V2):Z2=INT(V1*V3)
810 A1=P1*P2:A2=P1
820 A3=P1+P2-A1:A4=1:F=RND(0)
821 CLS:IF F<=A1 THEN 830
822 IF F>A1 AND F<=A2 THEN 870
823 IF F>A2 AND F<=A3 THEN 810
824 IF F>A3 AND F<=A4 THEN 950
830 PRINT"O TEMPO ESTA QUENTE E
 SECO"
840 D(1)=150+Z1*25:D(2)=300+Z2*
40:GOTO 970
870 PRINT"O TEMPO ESTA QUENTE E
 UMIDO"
880 D(1)=100+Z1*25:D(2)=200+Z2*
40:GOTO 970
910 PRINT"O TEMPO ESTA FRIO E S
ECO"
920 D(1)=250+Z1*25:D(2)=160+Z2*
40:GOTO 970
950 PRINT"O TEMPO ESTA FRIO E U
MIDO"
960 D(1)=200+Z1*25:D(2)=100+Z2*
40
970 PRINT"DEMANDA DE BATATAS AS
SADAS=";D(1)
980 PRINT"DEMANDA DE LATAS DE C
OCA=";D(2)
1000 PRINT"JOGADOR  GANHO  CUST
OS  LUCRO"
1010 GOSUB 1600
1020 NEXT K
1030 FOR I=1 TO 2000:NEXT
1090 CLS
1100 PRINT" RESULTADO FINAL APO
S 10 DIAS ":PRINT:PRINT
1110 PRINT"JOGADOR","LUCRO TOTA
L"
1120 FOR J=1 TO N
1130 PRINT J,TT(J):NEXT J
1140 PRINT:PRINT:PRINT"FIM DE J
OGO"
1150 END
```



```
1400 PRINT:PRINTTAB(5);"seu ped
ido por favor":PRINT
1410 FOR J=1 TO N
1420 PRINT"JOGADOR";J:PRINT
1430 INPUT"NUMERO DE BATATAS";O
(1,J)
1440 INPUT"NUMERO DE LATAS DE C
OCA COLA";O(2,J)
1450 NEXT J
1460 RETURN
1600 FOR J=1 TO N
1610 FOR I=1 TO 2
1620 L=O(I,J)
1630 IF D(I)<L THEN L=D(I)
1650 RV(I)=C2(I)*L
1670 TC(I)=C1(I)*O(I,J)
1680 IF D(I)<=L THEN RC(I)=TC(I
)-C3(I)*(O(I,J)-D(I))
1700 P(I)=RV(I)-TC(I)
1710 TP(I,J)=TP(I,J)+P(I)
1720 NEXT I
1730 TT(J)=TP(1,J)+TP(2,J)-200
1740 E=RV(1)+RV(2)
1750 C=TC(1)+TC(2)+20
1760 P=P(1)+P(2)-20
1770 PRINTUSING"   #     ###.##
####.## ###.##";J,E,C,P
1780 NEXT J
1790 RETURN
```

O programa focaliza os lucros e perdas de uma empresa do setor de alimentos, oferecendo ao usuário a opção de atuar sozinho ou de negociar com ate cinco outros "comerciantes".

O usuário também pode jogar no lugar de outras pessoas, tomando decisões diferentes conforme o papel assumido — o que lhe dá a oportunidade de comparar os resultados que obtém, agindo cautelosamente ou se arriscando em lances ousados. Tendo feito sua escolha, ele deve fornecer o número de participantes ao programa.

ELEMENTO DE ACASO

Cada jogador administra um bar que vende batatas assadas e latas de refrigerante. A demanda por um ou por outro produto depende do clima. Se está fa-

zendo frio, aumenta a procura pelas batatas; se faz calor, compra-se mais refrigerante. Infelizmente, o gerente precisa adquirir o estoque na véspera, desconhecendo, portanto, o tempo que vai fazer. Pode contar, até certo ponto, com o serviço de meteorologia, que costuma acertar, em média, 70% de suas previsões. Após dez dias de compras e vendas, o administrador que obtiver maior lucro vence.

PREVISÕES

A primeira parte do programa (até a linha 650) cuida das variáveis que controlam a tela e as condições de compra e venda. Você paga um aluguel de $ 20 por dia. As batatas custam $ 0,10 cada e são vendidas a $ 0,50. O refrigerante custa $ 0,15 para compra e $ 0,25 para venda. As sobras do estoque de um dia para outro são vendidas a um preço mais baixo — $ 0,01, as batatas e $ 0,12, o refrigerante. Cada partida dura dez dias.

O indicador da demanda de cada produto, conforme o clima, está na tabela da página 1185. Aqui entra um elemento importante do modelo. Naturalmente, os dias melhores para a venda de batatas assadas são os piores para a venda de refrigerantes e vice-versa. Contudo, há uma demanda suficiente para manter as vendas de ambos os produtos. Tudo depende de como aproveitamos as informações fornecidas pelo serviço de meteorologia, sempre tendo em mente, porém, que as previsões não são totalmente confiáveis. As probabilidades para um dia quente e seco são determinadas nas linhas 710 e 720, e depois utilizadas para simular as condições do tempo (linhas 810 a 824).

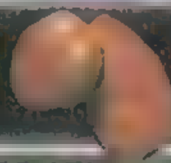
VARIÁVEIS ALEATÓRIAS

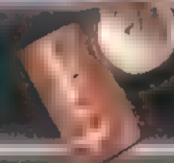
As linhas 770 a 800 contêm um método sofisticado para gerar variáveis aleatórias de distribuição normal. Elas são empregadas para simular a demanda na linha 840. A linha 770 cria duas variáveis aleatórias (**U1** e **U2**), processadas por meio de três fórmulas matemáticas. Na linha 780, **U1** é invertida e depois elevada ao quadrado; em seguida, calculamos o seu logaritmo natural e a raiz quadrada deste **V1**. A linha 790 faz **V2** igual ao co-seno de uma circunferência de raio U2, e **V3** igual ao seno dessa circunferência. **V1**, **V2** e **V3** sofrem um processamento adicional na linha 800 para resultar nas variáveis normais **Z1** e **Z2**.

As linhas restantes do programa cuidam da entrada e da impressão dos resultados na tela.

Para ser bem-sucedido neste jogo é preciso cuidar de cada centavo. Os gastos podem ser assustadoramente próximos dos ganhos — mesmo após dez dias de atividade.

INDICADOR DE DEMANDA

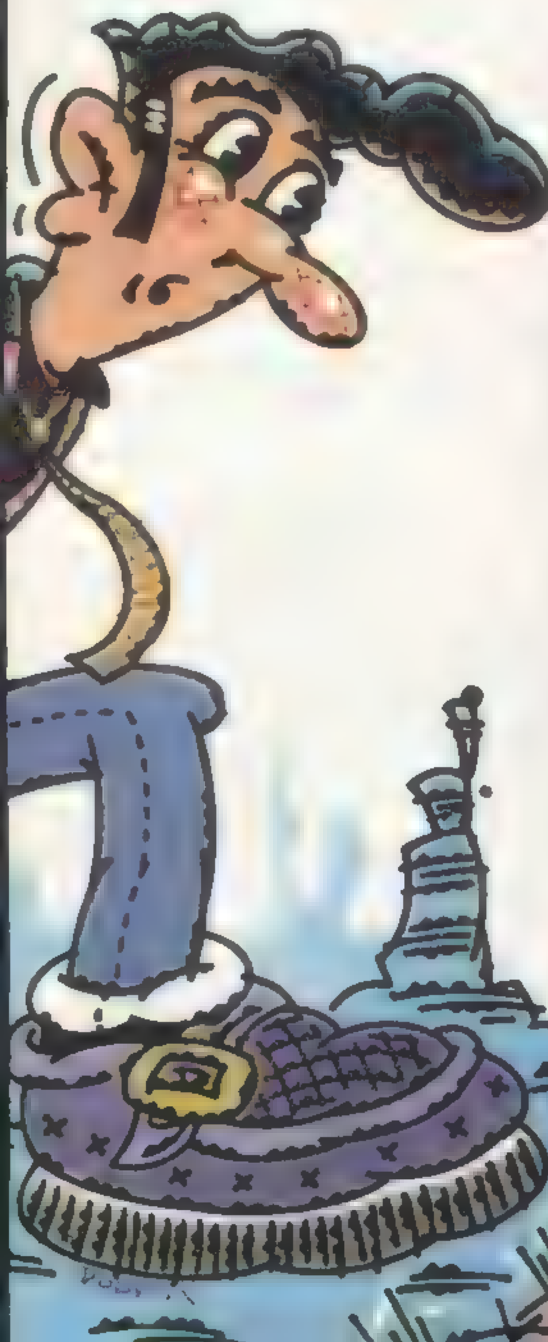




AVALANCHE: MAIS SALTOS

Até agora, Willie só podia pular verticalmente. A rotina deste artigo vai torná-lo capaz de saltar à frente, ultrapassando os obstáculos como um ágil cabrito montês.

Esta é a parte final da série de rotinas que cuidam dos movimentos de Willie. Finalmente, veremos nosso personagem não só subir a encosta e se desviar das pedras, como também saltar sobre os buracos e as cobras.



S

```
 10 REM org 59472
 20 REM mfj cp 129
 30 REM jr nz,mfb
 40 REM inc a
 50 REM ld (57335),a
 60 REM ld hl,(57332)
 70 REM ld de,22561
 80 REM add hl,de
 90 REM ld a,(hl)
100 REM cp 43
110 REM jp z,mdy
120 REM cp 44
130 REM jr nz,mfa
140 REM ld hl,(57332)
150 REM dec hl
160 REM ld (57332),hl
170 REM mfa ld hl,(57332)
180 REM ld de,32
190 REM sbc hl,de
200 REM ld (57332),hl
210 REM ld bc,57072
220 REM ld de,515
230 REM ld a,40
240 REM call 58970
250 REM ret
260 REM mfb cp 132
270 REM jr z,mfd
280 REM inc a
290 REM ld (57335),a
300 REM ld a,(57334)
310 REM cp 1
320 REM jr z,mfc
330 REM ld hl,(57332)
340 REM ld bc,16384
350 REM ld a,45
360 REM ld de,515
370 REM call 58970
380 REM inc hl
390 REM ld (57332),hl
400 REM ld bc,57000
410 REM ld a,40
420 REM ld de,258
430 REM call 58970
440 REM ld a,1
450 REM ld(57334),a
460 REM ret
470 REM mfc ld hl,(57332)
480 REM ld bc,57016
490 REM ld a,40
500 REM ld de,514
510 REM call 58970
520 REM inc hl
530 REM ld (57332),hl
540 REM ld de,22592
550 REM add hl,de
560 REM ld a,(hl)
570 REM cp 44
580 REM jr nz,mcf
590 REM ld a,0
600 REM ld (57335),a
610 REM ld a,3
620 REM ld b,5
630 REM call sci
640 REM mcf ld a,0
650 REM ld (57334),a
660 REM ret
670 REM mfd ld a,0
680 REM ld (57335),a
690 REM ld hl,(57332)
700 REM dec hl
710 REM ld bc,16384
720 REM ld a,45
730 REM ld de,514
740 REM call 58970
750 REM ld de,33
760 REM add hl,de
770 REM ld (57332),hl
780 REM jp 59153
790 REM org 59895
800 REM sci ret
810 REM org 59330
820 REM mdy *
```

- SAINDO DO CHÃO
- SALTO À FRENTE
- SUBIDA DA ENCOSTA
- TERRA FIRME
- CHECAGEM DA SEGURANÇA
- AVANÇANDO POSIÇÕES
- ALTERAÇÕES DO ESCORE
- AJUSTES FINAIS

Quando o programa chama esta rotina pela primeira vez, o acumulador contém 129. Esse valor é armazenado na posição 57335 da memória quando as teclas M e N são pressionadas juntas.

SAINDO DO CHÃO

Antes de mais nada, esta rotina verifica se o acumulador contém o valor 129. O número também poderia ser 130, 131 ou 132 e, se qualquer um deles estiver presente, os comandos **cp 129** e **jr nz,mfb** desviam o programa para o rótulo **mfb**. Na primeira vez que se chama a rotina, porém, o número no acumulador é 129.

A tarefa seguinte consiste em incrementar o acumulador e carregá-lo de volta em 57335. A instrução **ld hl,(57332)** transfere para a posição de Willie de 57332 para o par HL. O par DE é carregado com 22561 e adicionado em HL, para que obtenhamos a cor da posição adiante dos pés de Willie. A instrução **ld a,(hl)** carrega no acumulador o código dessa cor. Em seguida, o código é comparado com 43, a cor da cobra. Se esse valor é encontrado, Willie está saltando sobre a língua da cobra, e, fatalmente, leva uma picada e morre. A instrução **jp z,mdy** desvia então a rotina para o rótulo **mdy**, que providencia a eliminação de Willie.

Se ele não morreu, o programa compara a cor da posição adiante de seus pés com 44, a cor da terra. Se não há essa cor na frente de Willie, a instrução **jr nz,mfa** desvia o programa para a rotina **mfa**. Caso contrário, a posição de Willie é diminuída de um.

Em **mfa**, a posição de Willie é colocada em HL. DE é carregado com 32 e subtraído de HL, para mover o apontador um caractere acima na tela. Esse valor é colocado de volta em 57332.

O par BC é carregado com 57072, o endereço inicial para os dados de uma das figuras de Willie no salto para a frente. O par DE é carregado com 515, para especificar que um bloco dois por três será impresso — 2 x 256 + 3 = 515. A cor de Willie — 40, azul sobre ciano — é carregada com A, e a rotina de impressão é chamada.

Na próxima vez que ela for chamada, o acumulador conterá 130. Logo, o processador pulará a primeira parte, executando-a a partir do rótulo **mfb**.

Essa pequena rotina começa com um



teste. O conteúdo do acumulador é comparado com 132. Se esse valor está presente, a rotina **mfd** é chamada. Portanto, caso o acumulador contenha 130 ou 131, a rotina acionada é a **mfb**.

A primeira coisa que **mfb** faz é incrementar A e carregar o resultado de volta em 57335.

Em seguida, a variável da caminhada é carregada da posição de memória 57334 para o acumulador. Com isso, informa-se ao processador qual das duas figuras de Willie é necessária.

O conteúdo do acumulador é comparado com 1. Se 1 está presente, o programa salta para o rótulo **mfc** e imprime a figura de Willie com as pernas abertas. Antes que ela seja impressa, qualquer vestígio da figura anterior deve ser apagada. Para isso, a posição de Willie é transferida de 57332 para o par HL. O par BC é carregado com 16384, o início da tela. A é carregado com 45, para selecionar a cor ciano sobre ciano, e o par DE, com 515, para selecionar um bloco dois por três — 2 x 256 + 3 = 515. A rotina de impressão de bloco em 58970 é chamada, apagando a figura anterior de Willie.

HL é incrementado, movendo o apontador de tela para a próxima posição. Esse valor é colocado de volta no apontador do endereço 57332, ajustando-o também. BC é carregado com 57000, que é o início dos dados para a figura de Willie com as pernas juntas. A é ajustado com 40 — cor azul sobre ciano. DE é ajustado com 258 — o bloco de um por dois caracteres.

Impressa a figura de Willie, A é carregado com 1. Esse valor é armazenado em 57334, para que, na próxima vez, a outra figura seja impressa.

WILLIE AVANÇA

Quando a rotina for chamada novamente, a variável da caminhada em 57334 conterá 1 e o processador irá diretamente para o rótulo **mfc**. Nessa rotina, o par HL é carregado mais uma vez com a posição de Willie. BC é carregado com 57016, o endereço inicial para os dados da figura do personagem com as pernas abertas. A é ajustado de novo com 40, e DE, com 515 — a figura de Willie com as pernas abertas ocupa um bloco de dois por dois. A rotina de impressão é chamada, e Willie é impresso com as pernas abertas.

Observe que HL não foi incrementado nesta parte da rotina porque Willie ocupa, agora, duas posições. Ele avançou, portanto, aproximadamente metade de um caractere.

Uma vez que a figura tenha sido impressa, o apontador de tela em HL e 57332 é incrementado, fazendo com que Willie efetivamente avance.

SUBIDA DA ENCOSTA

Em seguida, devemos verificar se Willie está pisando na encosta ou não. DE é carregado com 22592, que é adicionado à posição de tela em HL. Com isso, ele passa a apontar para a cor da posição abaixo dos pés de Willie. A instrução **ld a,(hl)** carrega essa cor no acumulador, onde é comparada com 44. Se a cor da encosta não se encontra em A, Willie está no espaço aberto — e a instrução **jr nz,mcf** manda o processador para o rótulo **mcf**.

Mas, se o acumulador contém a cor da encosta, Willie conseguiu subir um aclive, merecendo um número adicional de pontos. É preciso, então, interromper o salto e atualizar o escore. Para isso, 0 é carregado em A e armazenado no endereço 57335. O valor 3 é carregado em A e 5 em B. Esses parâmetros serão utilizados por outra rotina, **sci** — ou incremento do escore. Na verdade, adicionaremos o valor 50 ao escore. O número 5 será adicionado à coluna das dezenas, que é a terceira a partir da esquerda. Em seguida, a rotina **sci** é chamada — como ela ainda não foi digitada, colocamos um **ret** provisório em seu endereço inicial.

Uma vez marcados os pontos, A é carregado com 0 e armazenado em 57334, quer Willie esteja pisando na encosta ou não. Indica-se, assim, que a próxima figura a ser impressa é a do personagem com as pernas juntas.

TERRA FIRME

Quando todas as figuras de Willie saltando já tiverem sido exibidas na tela, a variável de salto em 57335 terá o valor 132. Portanto, na próxima vez que a rotina de movimentação for chamada, o programa irá direto para a última rotina que começa com o rótulo **mfd**. Essa rotina começa igualando a variável de salto a 0. O valor 0 é carregado também em A e armazenado em 57335. A seguir, a posição de Willie na tela é decrementada.

O par BC é carregado com 16384, endereço do início da tela. A é igualado a 45, o código de ciano sobre ciano, e o par DE traz 514 para um bloco de dois por dois.

Chamando a rotina de impressão de bloco, apagaremos Willie da tela. O par DE é carregado com 33 e adicionado em HL. Com isso, movemos o apontador uma linha para baixo, colocando Willie de novo em terra firme. O resultado é armazenado em 57332.

Finalmente, a rotina de movimentação é acessada pela instrução **jp**. Essa rotina imprimirá Willie novamente de pé em sua posição sobre o solo, e pesquisará o teclado.

T

```
10  ORG 20321
20  MFJ CMPA #129
30  BNE MFB
40  INC 18261
50  LDX 18249
60  LEAX 290.X
70  LDA .X
80  CMPA #$57
90  LBEQ MDY
100  JSR MFZ
110  LDX 18249
120  LEAX -255.X
130  STX 18249
140  LDU #17814
150  JSR CHARPR
160  LEAX 254.X
170  LDU #17846
180  JSR CHARPR
190  RTS
200  MFB CMPA #130
210  BNE MFC
220  INC 18261
230  JSR MFZ
240  LDX 18249
250  LEAX -255.X
260  STX 18249
270  LDU #17870
280  JSR CHARPR
290  LEAX 254.X
300  JSR CHARPR
310  LEAX 254.X
320  JSR CHARPR
330  LDX 18249
340  LEAX 864.X
350  LDA .X+
360  CMPA #$FF
370  BEQ MFF
380  LDA .X
390  CMPA #$FF
400  BEQ MFF
410  RTS
420  MFF LDA #4
430  LDB #5
440  JSR SCI
450  RTS
460  MFC CMPA #131
470  BNE MFD
480  INC 18261
490  JSR MFZ
500  LEAX 254.X
510  LDU #1536
520  JSR CHARPR
530  LDX 18249
540  LEAX 257.X
550  STX 18249
560  LDU #17814
570  JSR CHARPR
580  LEAX 254.X
```

```
590  LDU #17846
600  JSR CHARPR
610  RTS
620  MFD CMPA #132
630  BNE MFE
640  JSR MFZ
650  LDX 18249
660  PSHS X
670  LEAX 512,X
680  LDY ,X
690  PULS X
700  CMPY #$5555
710  BNE MFE
720  LEAX 1,X
730  STX 18249
740  MFE CLR 18261
750  CLR 18251
760  LDX 18249
770  LDU #17774
780  JSR CHARPR
790  LEAX 254,X
800  JSR CHARPR
810  RTS
820  MFZ LDX 18249
830  LDU #1536
840  JSR CHARPR
850  LEAX 254,X
860  JSR CHARPR
870  RTS
880  CHARPR EQU 19402
890  SCI EQU 20751
900  MDY EQU 20126
```

Quando o processador executar esta rotina pela primeira vez, o acumulador conterá o número 129. Esse número é armazenado na posição de memória 18261 pela primeira parte da rotina de movimentação de Willie, quando M e N foram pressionadas simultaneamente.

PREPARAÇÃO DO SALTO

De início, a rotina verifica se o valor 129 se encontra no acumulador. Se em seu lugar estiverem os numeros 130, 131 ou 132, as instruções **CMPA #129** e **BNE MFB** desviam o programa para **MFB**. Na primeira vez que se chama esta rotina, porem, A contem 129.

O acumulador é incrementado e carregado de volta em 18261; em consequência, a segunda parte desta rotina será chamada na próxima vez.

A instrução **LDX 18249** transfere a posição de Willie de 18249 para X. O número 290 é adicionado para indicar o byte adiante dos pés do personagem. **LDA ,X** carrega o conteúdo desse byte no acumulador e a instrução **CMPA #$57** compara-o com $57, a cor gráfica para a língua da cobra. Se houver essa cor na frente de Willie no momento do salto, ele será picado e morrerá. Caberá à instrução **LBEQ MDY** comandar a execução da rotina da morte.

Se nosso personagem ainda está vivo, o processador vai para a rotina **MFZ**, que apaga os dois caracteres abaixo de Willie — para que os pés da figura anterior não permaneçam ali.

A posição de Willie, transferida outra vez de 18249 para X, é subtraída de 255, para mover a posição um caractere para cima. A nova posição na tela é armazenada de volta em 18249.

O registro U é então carregado com 17814, endereço inicial para os dados da figura com as pernas abertas. Lembre-se de que U é usado como apontador de dados pela rotina **CHARPR**. Essa rotina, que utiliza os dados como pilha do usuário, é chamada; X é somado a 254, movendo-se para a posição inicial da metade inferior de Willie. U é carregado com 17846, o início dos dados para a metade inferior da figura, e **CHARPR** é chamada mais uma vez, imprimindo Willie sobre a encosta.

SAINDO DO CHÃO

Quando a rotina de movimentação for chamada de novo, o acumulador conterá 130, e a rotina **MFB** será executada. Nas chamadas posteriores, o acumulador conterá um valor maior e as instruções **CMPA #130** e **BNE MFC** desviarão o fluxo de processamento para a próxima parte do programa. Como sempre, a primeira coisa que a rotina faz é incrementar o conteúdo de 18261. Depois, salta para **MFZ**, onde a metade inferior de Willie é apagada.

A posição seguinte do personagem é carregada em X e subtraída de 255 uma vez mais. Isso move o apontador de tela um caractere acima. O registrador U é carregado com 17870 e **CHARPR** é



chamada três vezes. Entre cada chamada, X é incrementado com 254. Imprime-se, assim, uma nova figura de Willie, em três caracteres. Embora sua altura permaneça a mesma, o personagem ocupa um caractere adicional porque foi deslocado meio caractere para cima.

SALVO DA PEDRA

Willie recebe mais pontos quando se desvia de uma pedra, pulando por cima dela. É necessário, portanto, verificar se há uma pedra sob a figura.

A posição de Willie é transferida de 18249 para X, sendo somada a 864, de modo a indicar a posição imediatamente abaixo. A é carregado com o valor do byte da tela apontado por X, e esse registrador é incrementado.

O byte da tela no registrador A é comparado com $FF, a cor gráfica da pedra. Se ela estiver presente, o processador vai para o rótulo **MFF**. Caso contrário, o próximo byte é carregado e comparado. Se a pedra estiver ali, o processador passa para o rótulo **MFF**; se não, ele retorna.

A rotina **MFF** atualiza os pontos. A é carregado com 4, para indicar o dígito que será ajustado — o quarto a partir da esquerda (dezenas). B é carregado com 5. A seguir, o programa salta para a rotina **SCI**, que tem a função de adicionar 50 pontos ao escore. Como ainda não digitamos essa rotina, convém colocar **RTS** em sua posição inicial, para evitar um erro no programa.

WILLIE AVANÇA

Na próxima vez que a rotina de movimentação é chamada, o acumulador contém 131. Caso esse valor não esteja presente, o programa é mandado para o rótulo **MFD**, pois o salto de Willie se encontra numa etapa mais adiantada.

Novamente, a primeira coisa que a rotina faz é incrementar a variável de salto em 18261. Quando a rotina voltar a ser chamada, o processador irá executar diretamente a parte seguinte.

Na sequência, o programa salta para a rotina **MFZ**, que apaga os caracteres superior e intermediário de Willie. Seus pés serão apagados com um caractere do céu pela próxima rotina. Esse trabalho de limpeza adicional precisou ser feito porque Willie tinha três caracteres de altura na última vez.

A posição em X, que foi ajustada durante a execução da rotina **MFZ**, é incrementada com 254, movendo o apontador para os pés de Willie. O apontador de dados em U é carregado com 1536, o endereço do céu limpo, no canto superior esquerdo da tela. **CHARPR** é chamada e imprime o caractere de céu.

A posição anterior de Willie é carregada de 18249 para X e adicionada a 257, movendo-se um caractere para baixo.



O apontador de dados em U é novamente carregado com o endereço inicial da figura de Willie com as pernas abertas em 17814. A rotina **CHARPR** imprime sua metade superior. O apontador de tela em X é incrementado, movendo-se para a linha de baixo. U é carregado com o endereço dos dados para a metade inferior da figura e **CHARPR** é chamada, realizando a impressão.

TERRA FIRME

O valor 132 no acumulador desvia o processador para a rotina que faz Willie pisar novamente em terra firme. Se não houver esse valor em A, o processador vai para o rótulo **MFE** — o que nunca deve ocorrer. Um conteúdo de A menor que 132 indica que o processador ficou executando alguma das rotinas anteriores. Isso acontece se a variável de salto em 18261 — endereço de onde vem o valor de A — ainda não foi incrementada. Temos, assim, um bom dispositivo de prevenção de erros: caso haja um valor incorreto em 18261, o processador salta diretamente para a instrução de limpeza, que reajusta esse endereço com 0.

Se o número em A é 132, o processador continua com esta rotina. Inicialmente, ele vai para a rotina **MFZ**, que apaga a última figura de Willie. Em seguida, a posição de Willie é carregada de 18249 para X. O apontador de tela

em X é guardado na pilha, e somado a 512, passando a apontar para o caractere imediatamente abaixo dos pés de Willie. Os dois bytes são carregados no registrador Y e o apontador de tela original é recuperado da pilha de volta para X.

O conteúdo de Y é comparado com $5555, o valor de um espaço amarelo na tela — ou seja, de um buraco. Se esse valor não estiver presente, o processador vai para o rótulo **MFE**

Se há um buraco alí, nosso personagem precisa de ajuda. X é então incrementado e colocado de volta em 18249, onde está o apontador de posição.

Como você notará, nenhuma verificação é feita quando Willie volta ao solo num nível mais alto. Depois de saltar e avançar, ele sempre aterrissa num nível mais alto.



AJUSTES FINAIS

O salto agora está completo. Precisamos, no entanto, limpar o contador de salto e a posição que contém o tipo de figura que deve ser impressa. Como essas posições são ajustadas com 0, o processador irá para a primeira parte da rotina de movimentação.

A seção do programa apresentada neste artigo finaliza imprimindo Willie de pé, em sua nova posição. X é carregado com a posição contida em 18249 e U, com a contida em 17774. A rotina **CHARPR** é chamada duas vezes. A última rotina, **MFZ**, é chamada para apagar a figura. X é carregado com a posição de Willie em 18249 e U é carregado com 1536, que equivale a um espaço vazio.

Para testar o programa, digite as seguintes linhas:

```
10 POKE 30000,57:POKE 20847,57:
POKE 20751,57
20 EXEC 19426
30 FOR L=1 TO 8:POKE 18261,129:
FOR J=1 TO 5:EXEC 19902
40 FOR K=1 TO 100:NEXT K,J,L
50 GOTO 50
```



O programa destinado ao MSX foi dividido em duas partes para facilitar a montagem.

```
10    org 55009
20    mp cp 129
30    jr nz,mb
40    inc a
50    ld (-5202),a
60    ld hl,(62407)
70    ld de,(-5205)
80    add hl,de
90    ld de,33
100   add hl,de
110   call 74
120   cp 36
130   jp z,54848
140   cp 52
150   jr nz,ma
160   ld hl,(-5205)
170   dec hl
180   ld (-5205),hl
190   ma ld hl,(-5205)
200   ld de,(62407)
210   add hl,de
220   push hl
230   ld de,32
240   add hl,de
250   ld a,255
260   push hl
270   call 77
280   pop hl
290   inc hl
300   ld a,255
310   call 77
320   pop hl
330   ld a,5
340   push hl
350   call 77
360   pop hl
370   inc hl
380   ld a,7
390   push hl
400   call 77
410   pop hl
420   ld de,32
430   sbc hl,de
440   ld a,6
450   push hl
460   call 77
470   pop hl
480   dec hl
490   ld a,4
500   call 77
510   ld hl,(-5205)
520   ld de,32
530   sbc hl,de
540   ld (-5205),hl
550   ret
560   mb cp 132
570   jp z,55275
580   inc a
590   ld (-5202),a
600   ld a,(-5203)
610   cp 1
620   jp z,55194
630   ld hl,(-5205)
640   ld de,(62407)
650   add hl,de
660   ld a,255
670   push hl
680   call 77
690   pop hl
700   ld de,32
710   add hl,de
720   ld a,255
730   push hl
740   call 77
750   pop hl
760   inc hl
770   ld a,1
780   push hl
790   call 77
800   pop hl
810   ld de,32
820   sbc hl,de
830   ld a,0
840   call 77
850   ld hl,(-5205)
860   inc hl
870   ld (-5205),hl
880   ld a,1
890   ld (-5203),a
900   ret
910   end
```

Complete agora o programa, digitando as seguintes linhas:

```
10    org 55194
20    mc ld hl,(-5205)
30    ld de,(62407)
40    add hl,de
50    ld a,4
60    push hl
70    call 77
80    pop hl
90    inc hl
100    ld a,6
110    push hl
120    call 77
130    pop hl
140    ld de,32
150    add hl,de
160    ld a,7
170    push hl
180    call 77
190    pop hl
200    dec hl
210    ld a,5
220    call 77
230    ld hl,(-5205)
240    inc hl
250    ld (-5205),hl
260    ld de,(62407)
270    add hl,de
280    ld de,64
290    add hl,de
300    call 74
310    cp 52
320    jr nz,ms
330    ld a,0
340    ld (-5202),a
350    ld a,3
360    ld b,5
370    call 57000
380    ms ld a,0
390    ld (-5203),a
400    ret
410    md ld a,0
420    ld (-5202),a
430    ld hl,(-5205)
440    ld de,(62407)
450    add hl,de
460    push hl
470    ld a,255
480    call 77
490    pop hl
500    dec hl
510    ld a,255
520    push hl
530    call 77
540    pop hl
550    ld de,32
560    add hl,de
570    ld a,255
580    push hl
590    call 77
600    pop hl
610    inc hl
620    ld a,255
630    call 77
640    ld hl,(-5205)
650    ld de,32
660    add hl,de
670    ld (-5205),hl
680    jp 54603
690    ret
700    end
```

Na primeira vez que o processador chama a rotina, o acumulador contém 129. Esse número é armazenado na posição de memória −5202 quando as teclas M e N são pressionadas juntas.

SAINDO DO CHÃO

Antes de mais nada, a rotina verifica se o valor que esta no acumulador é 129. O número em A poderia ser também 130, 131 ou 132. Caso um desses números esteja presente, as instruções **cp 129** e **jr nz,mb** desviam o programa para o rótulo **mb**. Porém, quando a rotina é chamada pela primeira vez, o número no acumulador é 129.

Depois dessa checagem, o acumulador é incrementado e carregado de volta em −5202. A seguir, o endereço inicial da Tabela de Nomes da VRAM é carregado em HL. A posição de Willie na tela é transferida de −5205 e −5204 para DE e somada em HL. Adiciona-se o valor 33 nesse par de registros através de DE. Com isso, HL passa a apontar para o endereço na TN ocupado pelo código do padrão que esta adiante dos pés de Willie. A rotina 74 da ROM é chamada, devolvendo a A o conteúdo da posição da VRAM apontada por HL — ou seja, ela faz a leitura na VRAM.

O código do padrão que se encontra adiante dos pés de Willie é comparado com 36, o código da língua da cobra. Se esse valor estiver presente, Willie fatalmente morrerá, pois está saltando sobre a língua da cobra e será picado. A instrução **jp z, 54848** manda então o processador para a rotina da morte, apresentada anteriormente com o rótulo **mre**. Se Willie não morreu, o processador compara o código do padrão à frente de seus pés com 52, que é o padrão da encosta. Se o valor não estiver presente, a instrução **jr nz,ma** desvia o programa para a rotina **ma**. Caso contrário, a posição do personagem em −5205 é decrementada.

A rotina **ma** coloca a posição de Willie em HL. DE é carregado com o endereço inicial da TN da VRAM e somado em HL. Esse valor é guardado na pilha. Em seguida, o número 32 é somado em HL através de DE, e o apontador passa a indicar os pés na figura anterior de Willie. O número 255, código do padrão de céu, é carregado em A. O apontador em HL é preservado na pilha e a rotina 77 da ROM é chamada, escrevendo o valor de A no endereço da VRAM apontado por HL. Apagamos, assim, os pés da figura anterior de Willie.

O último valor de HL é recuperado da pilha e incrementado. A é carregado com 255 e a rotina 77 é novamente chamada. Procedemos dessa maneira porque a posição de Willie poderia ter sido decrementada no começo da rotina — a operação nos dá a certeza de que os pés de Willie foram apagados.

O apontador inicial é recuperado da pilha. Em seguida, os padrões 4, 5, 6 e 7 são impressos, formando a figura de Willie com as pernas abertas. Acompanhe o apontador em HL e verifique como isso foi feito. A posição de Willie em −5205 e −5204 é atualizada, já que foi deslocada uma posição acima. Depois, o programa retorna partindo desse ponto da rotina.

Quando a rotina for chamada outra vez, o acumulador conterá 130. O processador pula, então, a primeira parte, indo para o rótulo **mb**.

Essa pequena rotina começa verificando o conteúdo do acumulador. Se ele contiver 132, a rotina **md** é chamada; se contiver 130 ou 131, o processador continua executando **mb**. A primeira coisa que essa rotina faz é incrementar A e carregar o resultado de volta em −5202. Em seguida, a variável da caminhada é carregada no acumulador. Isso indica ao processador qual das figuras de Willie é necessária.

O conteúdo do acumulador é comparado com 1. Se esse valor está presente, o processador vai para o rótulo **mc**, em 55194, e imprime a figura de Willie com as pernas abertas. Se a variável é 0, o processador imprime a figura de Willie com as pernas juntas.

Antes de imprimir qualquer uma das figuras, porém, devemos apagar todos os vestígios da imagem anterior. Para isso, a posição de Willie é transferida de −5205 e −5204 para HL. DE é carregado com o endereço inicial da TN da VRAM e somado em HL. A é carregado com 255, o código do padrão de céu. O apontador HL é guardado na pilha e a rotina 77 é chamada, apagando o quarto superior esquerdo da figura de Willie. O apontador é recuperado da pilha e somado com 32. A é carregado com 255. O apontador é guardado na pilha e a rotina 77 é chamada, apagando o quarto inferior esquerdo da figura. Com isso, apagamos a metade esquerda da antiga figura de Willie. A outra metade desaparece com a impressão da nova figura — desta vez, com as pernas abertas.

O apontador é recuperado da pilha e incrementado. A é carregado com 1, o código de um dos padrões da figura. O apontador é novamente colocado na pilha e a rotina 77 é chamada. Finalizando, o apontador é recuperado da pilha e subtraído de 32, movendo-se um

caractere para cima. A é carregado com 0, o código do outro padrão da figura. A rotina 77 é chamada e Willie é impresso com as pernas juntas.

A posição de Willie é transferida para HL, decrementada e devolvida em −5205 e −5204. O acumulador, carregado com 1, é armazenado em −5203, para que, na próxima vez, a outra figura de Willie seja impressa.

SALTO À DISTÂNCIA

Quando a rotina for chamada de novo, a variável da caminhada conterá 1 e o processador irá diretamente para o rótulo **mc**. Nessa rotina, a figura de Willie com as pernas abertas é impressa da mesma maneira que na rotina **ma**, só que sua posição foi incrementada — ou seja, Willie avança no ar!

Os padrões 4, 5, 6 e 7, que compõem a figura do personagem com as pernas abertas, são colocados na TN (na tela) pela rotina 77, que é chamada quatro vezes. Em seguida, a posição de Willie em −5205 e −5204 é colocada em HL, incrementada e devolvida aos endereços, o que faz a figura avançar mais uma posição.

SUBIDA DA ENCOSTA

Depois disso, devemos verificar se nosso personagem está pisando na encosta ou não. Lembre-se de que HL já contém a posição de Willie. DE é carregado com o endereço inicial da TN e somado em HL. O número 64 também é somado em HL através de DE. O apontador está indicando a posição sob os pés de Willie. A rotina 74 é chamada e faz a leitura da VRAM nessa posição. O valor é comparado com 52, o código da encosta. Se esse valor estiver presente, Willie subiu um lance na montanha. Para interromper o salto, coloca-se 0 em −5202. O número 3 é carregado em A e o 5, em B. Esses parâmetros serão utilizados por outra rotina, chamada na sequência, que realiza a contagem de pontos; o valor 50 será adicionado no escore. Mas, como ainda não digitamos essa rotina, convém colocar uma instrução **ret** provisória nesse endereço, o que podemos fazer por meio do comando BASIC **POKE 57000.201**.

Se o padrão da encosta não estiver presente, as intruções anteriores são ignoradas e o processador continua a partir do rótulo **ms**.

A é carregado com 0 e armazenado em −5203. Isso é feito independentemente de Willie estar pisando na encosta ou não, e indica que a próxima figura a ser impressa é a do personagem com as pernas juntas.

TERRA FIRME

Quando todas as outras figuras de Willie já tiverem sido impressas na tela, a variável do salto terá o valor 132. Assim, quando a rotina de movimentação for chamada novamente, o processador irá direto para a última rotina, que começa no rótulo **md**.

Em primeiro lugar, **md** ajusta a variável de salto com 0. Esse valor é carregado no acumulador e armazenado em −5202. Em seguida, a figura anterior de Willie com as pernas abertas é apagada — para isso, coloca-se o padrão 255 nas quatro posições, por meio da rotina 77. A posição de Willie é carregada em HL e somada com 32 através do par DE. Willie volta, com isso, a pisar em terra firme. O resultado é armazenado em −5205 e −5204.

Finalmente, a instrução **jp** chama a rotina de movimentação. Essa rotina, além de imprimir a figura de Willie de pé sobre o solo, pesquisará o teclado mais uma vez.



FIGURAS TRIDIMENSIONAIS

Com a técnica ensinada neste artigo, você poderá exibir na tela de seu micro diferentes objetos geométricos sólidos. Trace o perfil da figura escolhida e deixe a máquina fazer o resto por você.

Desenhar objetos em três dimensões, mesmo que simétricos, é uma tarefa difícil — mas não para quem tem um microcomputador. Com o programa apresentado neste artigo, tudo o que você tem a fazer é desenhar o perfil de um dos lados da figura. A máquina completará o trabalho, inclusive preenchendo as faces do sólido com traços que permitirão uma melhor visualização.

A principal função do programa é promover a rotação do perfil que você definiu em torno de um eixo central. Pode-se, portanto, desenhar qualquer figura que possua seções circulares — como vasos, garrafas, candelabros, copos, maçãs, laranjas, enfim, uma enorme variedade de objetos.

Como o programa *roda* a linha [illegible], a figura obtida é denominada *sólido de revolução*. As versões para todos os micros possibilitam que se veja o produto final de qualquer ângulo. As do Spectrum e do TRS-Color realizam uma pequena animação, girando a figura em torno de seu eixo.

ROTAÇÃO DA FIGURA

Para desenhar o perfil do sólido, movimentaremos uma linha na tela, fixando-a quando atingir uma posição considerada satisfatória. Essa técnica, já empregada em outros programas de *INPUT*, permite a visualização de cada uma das etapas do desenho.

Foi estabelecido um limite de vinte retas para a definição do perfil desejado. Na maioria dos casos, onde se utilizam apenas cinco ou seis retas, essa quantidade é mais do que suficiente. Porém, se você quiser traçar curvas, terá que construí-las com várias retas bem pequenas, podendo chegar perto daquele limite.

O ÂNGULO DE VISÃO

Uma grande vantagem do programa é permitir ao observador escolher o ângulo do qual quer ver a figura — não só de cima, de baixo ou de frente, mas de qualquer ângulo entre 0° e 180° (um ângulo de 70°, por exemplo, mostra o objeto como se ele estivesse sobre uma mesa). Você pode mudar esse ângulo quantas vezes quiser e o micro redesenhará o sólido em outra posição.

O programa armazena as coordenadas de todas as linhas. Quando você acaba o desenho, ele faz a rotação de cada linha ao redor do eixo central, de 18 em 18°. Executa em vinte etapas a rotação completa. O ângulo de visão já é levado em conta nesta fase; os círculos aparecem mais achatados à medida que nos aproximamos dos 90°.

Assim que o desenho fica pronto, o computador espera que o usuário pressione a barra de espaços para escolher um outro ângulo.

A ANIMAÇÃO

No Spectrum e também no TRS-Color, você poderá ver o objeto executar a rotação, ao pressionar uma outra tecla. Para isso, o computador desenha as posições intermediárias da rotação e as armazena na memória, recorrendo à técnica de páginas gráficas. Depois que todas as páginas estão feitas, elas são exibidas em seqüência, o que produz o efeito de animação.

UTILIZAÇÃO DO PROGRAMA

Digite o programa e tente traçar algumas figuras sólidas. O processo de desenho consiste em movimentar o cursor para a posição desejada e marcar [illegible]



- DESENHOS EM TRÊS DIMENSÕES
- CRIE ALGUMAS FIGURAS
- CONSTRUÇÃO DO PERFIL
- COMO ALTERAR
- O ÂNGULO DE VISÃO
- ROTAÇÃO DA FIGURA
- COMO FUNCIONA O PROGRAMA
- AS TRÊS ROTINAS PRINCIPAIS

[illegible] colocado para qualquer lugar. Quand[illegible] [illegible] com sua localiz[illegible] [illegible] para fixá-la. Pres[illegible] [illegible] o perfil da figu[illegible] [illegible] o desenho [illegible] outra tecla.

[illegible] linha, M para [illegible] <ENTER> [illegible] Q [illegible] nar o desenho.

No TRS-Color [illegible] <ENTER> [illegible]

No MSX [illegible] barra de espaço para fixar [illegible] para finalizar. As teclas R e D [illegible] mais devagar.

No Apple [illegible] movimentar a linha. Os outros [illegible] MSX.



```
[illegible] PRINT AT 0,11; INK [illegible]
PAPER [illegible] "ROTACAO 3D" [illegible]
100
[illegible]
[illegible] CLEAR [illegible]
[illegible] CLS : POKE 23658,0
40 GOSUB 2000: BORDER 0
[illegible] PRINT AT 0,0; PAPER [illegible]
0;" MOVER LINHA - TECLAS DE C
URSOR [illegible]
[illegible]
[illegible] FIXAR LINHA - ENTER
[illegible] FIXAR CURSOR -
M    FIM - Q
[illegible] INPUT "ANGULO DE OBSERVACA
O (0-180)" [illegible]
THEN GOTO [illegible]
[illegible] PLOT 60,[illegible]: DRAW 135,0
DRAW 0,-49: DRAW -135,0: DRAW
0,49
[illegible]
[illegible]
[illegible]
120 LET xx=x: LET yy=y
130 OVER [illegible]
140 PLOT [illegible]: DRAW xx-x,yy-[illegible]
150 PLOT x,y: DRAW xx-x,yy-[illegible]
[illegible]
170 LET z=CODE [illegible]
180 IF z=13 THEN GOSUB 500
190 IF z=113 THEN GOTO 600
195 IF z=109 AND bf<>1 THEN
[illegible] 255-x,[illegible]
yy: LET bf=1
200 IF z=53 AND xx>128 THEN
LET xx=xx-1
210 IF z=8 [illegible]
LET xx=xx-2
220 IF z=5[illegible]
LET yy=yy+[illegible]
230 IF z=11 AND yy=98 THEN
LET yy=yy+2
240 IF z=56 [illegible]
LET xx=xx+1
250 IF z=9 [illegible]
LET xx=xx+[illegible]
260 IF z=54 [illegible]
LET yy=yy-[illegible]
270 IF z=10 AND yy>51 THEN
LET yy=yy-2
280 GOTO 140
500 OVER 0: PLOT x,y: DRAW xx-
x,yy-y
510 PLOT 255-x,y: DRAW x-xx,yy
-y
520 LET x=xx: LET y=yy: LET a$
=a$+CHR$ (x-128)+CHR$ (y-51)
OVER 1: RETURN
600 INK 7: CLS : OVER 0: DIM a
(20,2)
610 FOR a=0 TO [illegible]
620 PRINT AT 21,7; PAPER 2;
BRIGHT 1;"DESENHANDO FIGURA"
a+1
630 GOSUB 1000
640 IF a<>0 THEN GOTO 670
650 PRINT AT 19,0; INVERSE 1;"
<SPACE> P/ ALTERAR ANGULO DE
OBSERVACAO - QUALQUER OUT
RA TECLA PARA CONTINUAR"
660 LET X$=INKEY$: IF X$="" 
THEN GOTO 650
662 IF X$<>" " THEN GOTO 670
664 INPUT "DIGITE O NOVO ANGUL
O (0-180)" [illegible]
666 [illegible] THEN GOTO
662
668 CLS : LET [illegible]=SIN ([illegible]/180*PI
[illegible] GOTO 620
670 GOSUB [illegible]
680 CLS
690 NEXT [illegible]
700 PRINT AT [illegible] PAPER [illegible]
INK 1;" FIGURA: [illegible]
710 FOR a=128 TO 240 STEP 16
720 POKE 30114,[illegible]: RAND USR
30112
730 [illegible]
740 NEXT a: GOTO 710
1000 FOR b=1 TO LEN a$ STEP 2
1010 LET x=CODE a$(b)
1020 LET y=CODE a$(b+1)
1030 GOSUB 1500
1040 NEXT b
1050 RETURN
1500 FOR c=0 TO 399 STEP 20
1510 LET d=c+(2.25*a): LET py=[illegible]
e*y
1515 IF d>=360 THEN LET d=d-36
0
1520 GOSUB 1530: NEXT [illegible]
```

```
1520 DRAW bx-xd,by-yd: RETURN
1530 LET yd=SIN (d/180*PI)*x*COS (i/180*PI)
1532 LET xd=COS (d/180*PI)*x
1535 LET xd=128+xd: LET yd=90+yd+py
1540 IF c=0 THEN PLOT xd,yd: LET bx=xd: LET by=yd
1550 DRAW xd-PEEK 23677,yd-PEEK 23678
1560 IF [illegible] THEN GOTO 1580
1565 IF b=1 THEN PLOT 128,90: DRAW xd-128,yd-90: GOTO 1580
1570 PLOT a(1+c/20,1),a(1+c/20,2): DRAW xd-a(1+c/20,1),yd-a(1+c/20,2)
1580 LET a(1+c/20,1)=xd
1585 LET a(1+c/20,2)=yd
1590 RETURN
1600 POKE 30102,128+a*16
1610 RAND USR 30100
1620 RETURN
2000 RESTORE 2050
2010 FOR a=0 TO 23
2020 READ b: POKE a+30100,b
2030 NEXT a
2040 RETURN
2050 DATA 17,0,0,33,0,64,1,0,16,237,176,201
2060 DATA 33,0,0,17,0,64,1,0,16,237,176,201
```

```
10 PMODE 3,1:PCLEAR [illegible]:CLEAR [illegible],7679:DIM A(19,1)
20 RD=ATN(1)/45:V=24[illegible]
30 PCLS 1:SCREEN 1,0
40 BX=128:BY=190:DRAW"BM128,190[illegible]":X=BX:Y=BY
50 LINE [illegible]
60 LINE (BX,BY)-(X,Y),PRESET
70 IF PEEK(345)=V GOSUB 500
80 IF PEEK(339)=191 THEN M=4 ELSE M=1
90 LINE [illegible]
[illegible]EEK(338)[illegible] AND A$[illegible]X:BY=Y:BF=1:GOSUB 500
100 IF PEEK(339)[illegible] THEN 160
110 IF PEEK([illegible]) THEN Y=Y-M
120 IF PEEK([illegible]) THEN Y=Y+M
130 IF PEEK([illegible]) THEN X=X-M
140 IF PEEK(344)=V AND X+M<223 THEN X=X+M
150 GOTO 60
160 GOSUB 2000:SCREEN [illegible]:FOR A=0 TO 6:PCLS1:FOR G=0 TO A:PRESET(0,G*2):NEXT
170 GOSUB 1000
180 G$=INKEY$
190 IF A=0 AND G$[illegible] THEN 180
200 IF A=6 AND G$[illegible] THEN 160
210 NEXT
220 FOR A=0 TO 6:PCOPY [illegible] TO 1:PCOPY 6+A*2 TO 2:NEXT
230 IF INKEY$[illegible] THEN 160 [illegible] 220
500 LINE(BX,BY)-(X,Y),PRESET
510 LINE(255-BX,BY)-(255-X,Y),PRESET
520 BX=X:BY=Y:A$=A$+CHR$([illegible])+CHR$(190-Y):RETURN
1000 FOR B=1 TO LEN(A$) STEP 2
1010 X=ASC(MID$(A$,B,1)):Y=ASC(MID$(A$,B+1,1))
1020 GOSUB 1500:NEXT
1030 PCOPY 1 TO 5+A*2:PCOPY 2 TO 6+A*2
1040 RETURN
1500 FOR C=0 TO 360 STEP 20
1510 D=C+20*A/7:PY=18*Y[illegible]
```

```
[illegible] GOSUB [illegible]:NEXT:RETURN
1530 YD=110-SIN(D*RD)[illegible]
[illegible] XD=128+COS(D*RD)[illegible]
[illegible] IF C=0 THEN LINE(XD,[illegible])
[illegible] PRESET:BX=XD:BY=YD
1550 LINE-(XD,YD),PRESET
1560 IF B=1 AND BF=1 THEN 1580
[illegible] IF B=1 THEN LINE(128,[illegible])-(XD,YD),PRESET:GOTO 1580
[illegible] LINE(A(C/[illegible])-(XD,YD),PRESET
[illegible]
RETURN
2000 [illegible]
[illegible]
```

```
10 SCREEN 2:DIM A(1[illegible]
20 RD=ATN(1)/45:M=1
30 LINE (127,[illegible]
35 LINE (47,0)[illegible]
40 BX=127:BY=126:X=BX:Y=BY:XX=BX:YY=BY
60 LINE (BX,BY)-(X,Y),8
70 C$=INKEY$:IF C$="" THEN [illegible]
75 IF C$=" " THEN GOSUB [illegible]
80 IF C$="R" THEN M=4
85 IF C$="D" THEN M=1
90 LINE(BX,BY)-(X,Y),15:IF C$=" " AND A$="" THEN BX=X:BY=Y:BF=1:GOSUB 500
100 IF C$=[illegible] THEN 160
110 IF C$=CHR$([illegible]) AND [illegible] THEN Y=Y-M
120 IF C$=CHR$([illegible]) AND [illegible] THEN Y=Y+M
130 IF C$=CHR$([illegible]) AND [illegible] THEN X=X-M
140 IF C$=CHR$(28)AND X+M<207 THEN X=X+M
150 GOTO 60
160 GOSUB 2000
170 COLOR [illegible]
```

```
1000
190 C$=INKEY$:IF C$="" THEN 190
200 GOTO 160
500 LINE (BX,BY)-(X,Y),1
510 LINE (255-BX,BY)-(255-X,Y),
1
520 BX=X:BY=Y:A$=A$+CHR$(X-126)
+CHR$(127-Y):RETURN
1000 FOR B=1 TO LEN(A$) STEP 2
1010 X=ASC(MID$(A$,B,1)):Y=ASC(
MID$(A$,B+1,1))
1020 GOSUB 1500:NEXT B
1040 RETURN
1500 FOR C=0 TO 399 STEP 20
1510 D=C:PY=IS*(Y*.5)
1520 GOSUB 1530:NEXT C:RETURN
1530 YD=96-SIN(D*RD)*X*COS(I*RD
)-PY:XD=127+COS(D*RD)*X
1540 IF C=0 THEN BX=XD:BY=YD:XX
=XD:YY=YD
1550 LINE(XX,YY)-(XD,YD),1:XX=X
D:YY=YD
1560 IF B=1 AND BF=1 THEN 1580
1565 IF B=1 THEN LINE(80,91)-(X
D,YD),1:GOTO 1580
1570 LINE(A(C/20,0),A(C/20,1))-
(XD,YD),1
1580 XX=XD:YY=YD:A(C/20,0)=XD:A
(C/20,1)=YD:RETURN
2000 SCREEN 0:KEY OFF:COLOR 15,
4,4:LOCATE 8,8 :INPUT"Angulo de
 visão:";I
2010 IF I<0 OR I>180 THEN 2000
2020 IS=SIN(I*RD):RETURN
```



```
10  HOME : HGR : DIM A(19,1)
20 RD =  ATN (1) / 45:M = 1
30  HCOLOR= 6: HPLOT 140,90 TO
220,90 TO 220,149
40 BX = 140:BY = 149:X = BX:Y =
 BY:XX = BX:YY = BY
50  VTAB (21): PRINT "COMANDOS:
 <I>NICIO-<F>IM
52  VTAB (22): PRINT "VELOCIDAD
E DO RISCO: <R>APIDO-<D>EVAGAR"

54  VTAB (23): PRINT "BARRA DE
ESPACO: FIXA A RETA"
56  VTAB (24): PRINT "DIRECAO:
<Z>,<X>,<;>,<.>";
60  HCOLOR= 3: HPLOT BX,BY TO X
,Y
70  GET C$: IF C$ = " " THEN  G
OSUB 500
80  IF C$ = "R" THEN M = 4
85  IF C$ = "D" THEN M = 1
90  HCOLOR= 0: HPLOT BX,BY TO X
,Y: IF C$ = "I" AND A$ = "" THE
N BX = X:BY = Y:BF = 1: GOSUB 5
00
100  IF C$ = "F" AND  LEN (A$)
> 0 THEN 160
110  IF C$ = ";" AND Y - M > 90
 THEN Y = Y - M
120  IF C$ = "." AND Y + M < 15
0 THEN Y = Y + M
130  IF C$ = "Z" AND X - M > 14
0 THEN X = X - M
140  IF C$ = "X" AND X + M < 22
0 THEN X = X + M
150  GOTO 60
160  HGR : GOSUB 2000
170  GOSUB 1000
190  GET C$: IF C$ = " " THEN 1
60
195  GOTO 190
500  HCOLOR= 3: HPLOT BX,BY TO
X,Y
510  HPLOT 281 - BX,BY TO 281 -
 X,Y
520 BX = X:BY = Y:A$ = A$ +  CH
R$ (X - 140) +  CHR$ (149 - Y):
 RETURN
1000  FOR B = 1 TO  LEN (A$) ST
EP 2
1010 X =  ASC ( MID$ (A$,B,1)):
Y =  ASC ( MID$ (A$,B + 1,1))
1020  GOSUB 1500: NEXT B
1040  RETURN
1500  FOR C = 0 TO 399 STEP 20
1510 D = C:PY = IS * (Y * .5)
1520  GOSUB 1530: NEXT C: RETUR
N
1530 YD = 80 -  SIN (D * RD) *
X *  COS (I * RD) - PY:XD = 140
 +  COS (D * RD) * X
```



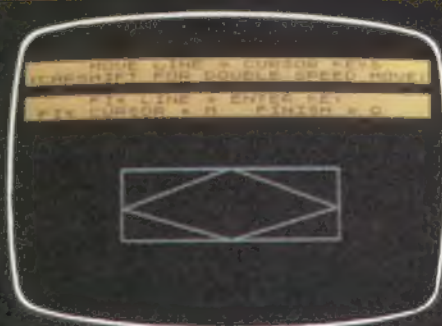
É possível realizar em um micro animações gráficas em três dimensões como as que aparecem na TV?

Os grafismos animados em três dimensões que aparecem em vinhetas de televisão, e nos filmes produzidos por computador (*TRON*, por exemplo), devem sua espantosa qualidade e realismo às máquinas que os elaboram — geralmente, supercomputadores do tipo Cray II, com capacidade de executar sessenta milhões de operações em ponto flutuante por segundo.

A técnica utilizada, entretanto, não difere da que explicamos neste artigo para gerar sólidos de revolução. Superfícies complexas são montadas por milhares de "tijolos" regulares, depois, "alisados" por algoritmos especiais, o que lhes dá um aspecto contínuo. Outros programas pintam as superfícies com cores selecionadas, adicionam brilho e sombra etc., tornando o resultado o mais realista possível.

Os micros podem, teoricamente, executar quase tudo o que um computador de grande porte faz. Porém, têm telas com menor resolução gráfica, menor número de cores, enfim, não contam com os mesmos recursos. Não se pode esperar, portanto, que produzam efeitos de igual qualidade.





A figura tridimensional é construída a partir de um simples perfil.



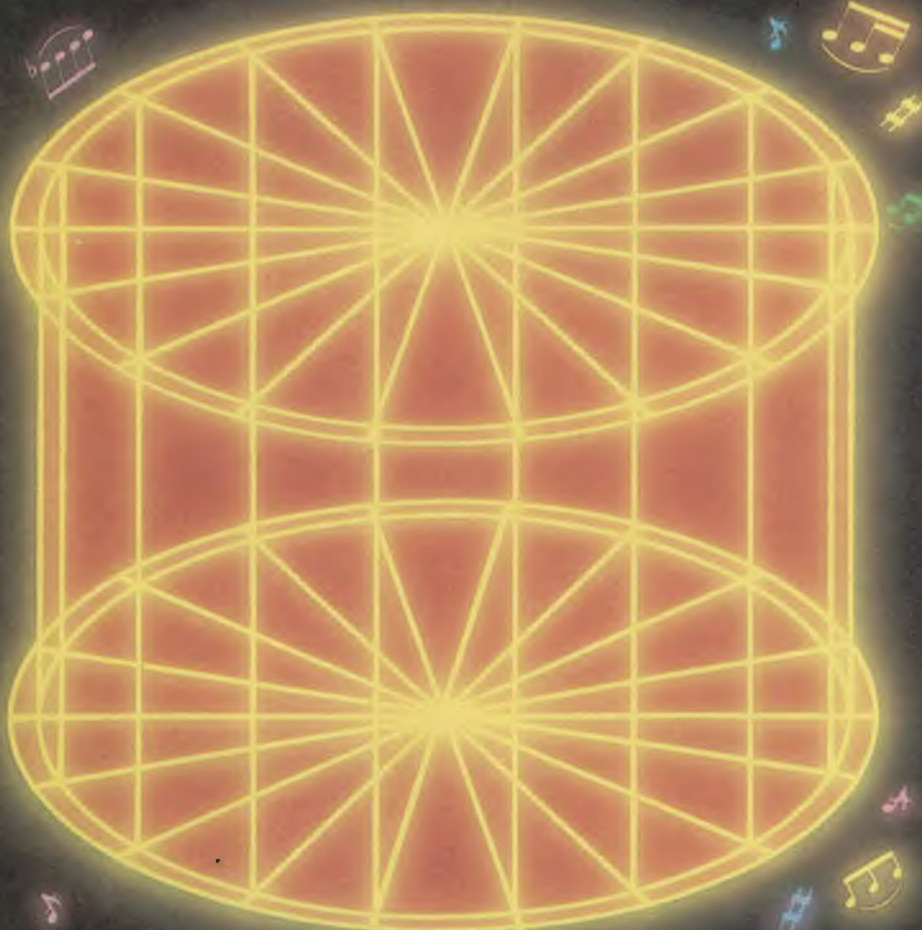
O microcomputador elabora o desenho executando a rotação de cada reta.



O produto final — o *sólido de revolução* — é exibido em três dimensões.

```
1540  IF C = 0 THEN BX = XD:BY
= YD:XX = XD:YY = YD
1550  HCOLOR= 3: HPLOT XX,YY TO
 XD,YD:XX = XD:YY = YD
1560  IF B = 1 AND BF = 1 THEN
1580
1565  IF B = 1 THEN  HPLOT 80,9
1 TO XD,YD: GOTO 1580
1570  HPLOT A(C / 20,0),A(C / 2
0,1) TO XD,YD
1580 XX = XD:YY = YD:A(C / 20,0
) = XD:A(C / 20,1) = YD: RETURN
```

```
2000  HOME : VTAB (23): INPUT "
ANGULO DE VISAO (0-180): ";I
2010  IF I < 0 OR I > 180 THEN
2000
2020 IS =  SIN (I * RD): RETURN
```



COMO FUNCIONA

Todos os programas funcionam de modo muito semelhante. As partes mais importantes são as rotinas que permitem o traçado do perfil, criam o sólido de revolução e fazem-no girar.

INTRODUÇÃO DO PERFIL

A rotina de desenho começa na linha 60 (linha 140, no micro Spectrum). Ela simplesmente lê as teclas que foram pressionadas e atualiza o valor das coordenadas da linha auxiliar. O desenho definitivo é feito pela rotina da linha 500, que também traça o perfil esquerdo da figura.

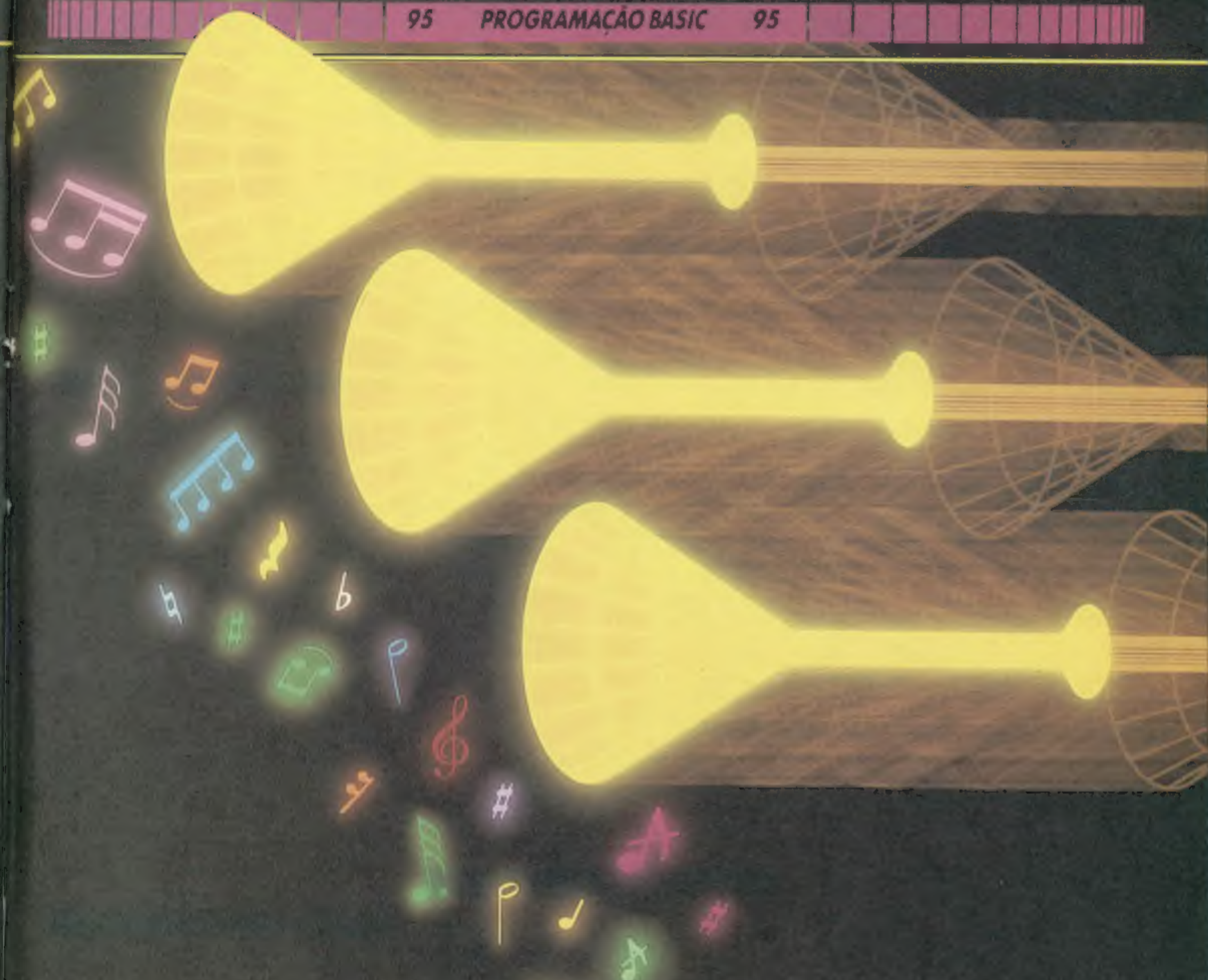
Cada vez que se fixa uma linha, as coordenadas de sua extremidade são armazenadas. Primeiro, subtrai-se um número de cada coordenada para obter uma nova coordenada, relacionada ao ponto de partida do desenho. Em seguida, esses números são convertidos em caracteres e adicionados ao final de uma variável alfanumérica (**A$** ou **a$**).

A rotina da linha 2000 (70, no Spectrum) permite que se defina o ângulo de visão. Depois, o programa é desviado para a linha 1000, que cria e desenha o sólido em três dimensões.

TERCEIRA DIMENSÃO

Essa rotina utiliza as coordenadas previamente armazenadas e o ângulo de visão que você definiu para transformar o perfil em um sólido de revolução. A técnica matemática usada por esta rotina é a mesma para todos os computadores, embora os comandos gráficos sejam diferentes.

Na linha 1000, um laço controlado pela variável B seleciona cada uma das suas linhas, tomando as coordenadas finais da variável **A$**. O programa salta, então, para a rotina da linha 1500. Essa rotina desloca as coordenadas finais de acordo com o ângulo requerido e traça as novas retas que irão compor a fa-



ce do sólido relativa àquela reta inicial. Isso é feito vinte vezes — sua linha inicial será, portanto, redesenhada vinte vezes ao redor de um círculo. Em seguida, a rotina retorna à linha 1000 para fazer o mesmo com a próxima linha. Ao executar o programa, você poderá observar essas etapas da elaboração do sólido.

Como você já deve ter observado, o processo empregado para desenhar a primeira linha da figura é um pouco diferente do utilizado para as demais, já que esta linha possui duas coordenadas, a inicial e a final.

As linhas 1560 e 1565 verificam se a reta a ser trabalhada é a inicial. Se **B** = 1, a reta é realmente a primeira de todas; se **BF** = 1, ela não deve ser traçada, pois corresponde apenas ao deslocamento para a definição do ponto inicial. Se houve um deslocamento, o programa armazenará as coordenadas do ponto inicial na variável alfanumérica. Essas coordenadas serão o ponto de partida para a próxima linha. Caso o primeiro movimento não tenha sido um deslocamento, o ponto inicial será o centro, a posição inicial do cursor; o ponto fixado, por sua vez, será a coordenada final da primeira reta.

Através deste processo, cada reta do perfil será "multiplicada" por vinte e todas elas terão suas extremidades ligadas por um círculo, gerando o efeito de três dimensões.

A ROTAÇÃO

O Spectrum e o TRS-Color, como dissemos, ainda oferecem o artifício de animar a figura na tela, ao toque de uma tecla. O número de figuras (**A**) é controlado pelo laço das linhas 610 a 690, no Spectrum, e da linha 220 no TRS-Color. A variável **A** controla também a rotação parcial de cada figura, por meio de uma modificação no ângulo de visão (linha 1510). Finalmente, os quadros são exibidos em seqüência, criando o efeito de animação.

VIDEOTEXTO E MICROCOMPUTADORES

Use o micro para ter acesso a um sofisticado sistema de intercâmbio de informações: o videotexto. Ele colocará à sua disposição uma variada gama de serviços.

Utilizado isoladamente, um microcomputador doméstico é capaz de desempenhar uma grande variedade de tarefas. Estas se multiplicam, porém, quando ele passa a integrar uma *rede de computadores* — o que, nos últimos anos, se tornou possível no Brasil.

No artigo da página 561, vimos como a tecnologia dos *modems* (modulador-demodulador) permite que se recorra à rede telefônica normal para envio e recepção de mensagens, textos, dados, programas e gráficos em forma digital. Um computador pode, assim, "conversar" com outro de uma maneira barata e relativamente rápida.

Existem diferentes tipos de rede de computadores que empregam o telefone e o modem como meios de comunicação. As redes de área local (LAN) ou ampla (WAN), compostas apenas por microcomputadores (como os quadros de avisos, ou *bulletins boards*), serão examinadas em outro artigo. Aqui, trataremos das redes em que o microcomputador é usado como terminal de um computador maior.

COMO FUNCIONA

O videotexto é um exemplo desse tipo de rede. O sistema estabelece o contato do usuário com um computador de grande porte, por meio da rede telefônica normal. O microcomputador não é indispensável à rede: o usuário precisa ter apenas um terminal de videotexto. Este é um pequeno aparelho dotado de microprocessador e teclado portátil, que usa como saída de vídeo um aparelho de TV doméstico.

O software operacional do videotexto permite não só a implementação de uma estrutura própria de dados, bem como sua consulta pelo usuário. As informações armazenadas no computador central do sistema videotexto são organizadas em *páginas*. Cada página corresponde a uma ou mais telas de vídeo contendo texto e ilustrações, ambos em cores. Para sua composição, utiliza-se um conjunto de caracteres gráficos típicos de videotexto.

As páginas são organizadas segundo uma estrutura em forma de árvore. O usuário pode "explorar" à vontade essa árvore, chamando à tela — por meio de seleções efetuadas pelo teclado — as diversas páginas que compõem um determinado conjunto. As opções feitas são enviadas ao computador central, que então encaminha a página solicitada para o terminal de videotexto, por intermédio da linha telefônica.

O uso do videotexto é aberto a assinantes — ou seja, o usuário que desejar ter acesso ao mesmo deve pagar uma assinatura mensal, debitada na conta telefônica normal, além de uma taxa de utilização, que varia proporcionalmente ao tempo que o terminal permanece conectado ao sistema.

Os dados armazenados no computador central são fornecidos por empresas, órgãos governamentais, jornais, revistas e outras fontes, que mantêm suas próprias páginas de informação, atualizando-as freqüentemente.

O videotexto oferece uma enorme variedade de serviços informativos e recreativos: notícias do dia, indicadores econômicos e cotações das bolsas, programação de cinemas e teatros, publicações, transportes aéreos, turismo, informações científicas e médicas, horóscopo, números da lista telefônica, produtos à venda etc. Como o sistema é interativo, inclui ainda serviços em que a participação do usuário é mais intensa, tais como: videogames, troca de mensagens pessoais, anúncios classificados, reserva de passagens aéreas, solicitação de catálogos, concursos, telecompras etc.

O sistema é fácil de usar, fornecendo ao usuário todas as instruções necessárias e os menus para seleção.

CONEXÃO A UM MICRO

Como já dissemos anteriormente, mesmo aqueles que não possuem um computador doméstico podem se associar ao sistema videotexto utilizando um terminal alugado pela própria companhia telefônica local por um valor módico (não é necessário comprá-lo).

Entretanto, a conexão de um micro ao sistema videotexto traz algumas vantagens ao usuário — entre elas o acesso a alguns serviços de fornecimento gratuito de software, que pode ser copiado com grande facilidade em fita cassete ou disco. Dispondo de um computador, o usuário tem ainda a possibilidade de listar em impressora as informações fornecidas pelo sistema, para exame posterior.

Para conectar o seu micro ao sistema videotexto, basta ter:

- uma porta serial assíncrona, do tipo RS-232C (se seu computador não dispõe de uma, você precisará comprar uma placa de expansão);

- um modem de 1200 bips por 75 (existem tipos especiais para videotexto, alguns deles disponíveis também como placas internas de expansão);

- um programa de emulação de terminal de videotexto, que deve ser carregado previamente na memória do seu micro, quando quiser acessar o serviço.

Já existem "pacotes" para acesso ao videotexto, contendo os componentes mencionados, para praticamente todos os tipos de microcomputadores nacionais. Algumas companhias telefônicas alugam ou vendem *kits* de acesso para os modelos mais populares, como o Apple II e seus compatíveis. Mesmo micros bem simples, como o TK-90X, podem ser conectados ao videotexto (não é preciso ter uma unidade de disquete).

Finalmente, para se beneficiar do sistema, será necessário que você se inscreva como assinante do videotexto junto à companhia telefônica. Consulte-a para saber se o serviço está disponível em sua cidade e se seu micro foi credenciado para acesso ao videotexto. Em caso de resposta afirmativa, você poderá obter também a informação sobre onde conseguir o *kit* de acesso, para aluguel ou compra.